Analytic Interpolation and the Non-linear Fourier Transform

Julián Uribe Jaramillo

Master of Science Thesis

Delft Center for Systems and Control

Analytic Interpolation and the Non-linear Fourier Transform

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Julián Uribe Jaramillo

May 25, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

The non-linear Fourier transform may be considered an extension of Fourier analysis to non-linear problems. It has applications in many different scientific and engineering disciplines, for instance as a solution to the non-linear Schrödinger equation which describes how electromagnetic waves travel through ideal non-linear media. Hence, the non-linear Fourier transform is fundamental for communication schemes that take optical fiber non-linearities into account.

Wahls et al. proposed an algorithm for a fast inverse non-linear Fourier transform based around a discretization of the non-linear Schrödinger equation. But one of the steps in the algorithm — known as synthesis — only had heuristic solutions. However, the problem closely resembles a Nevanlinna-Pick interpolation with degree constraint, a problem known to have a guaranteed solution. Indeed, synthesis is a limit case of Nevanlinna-Pick interpolation with degree constraint on the boundary of the region of analyticity. Although there are solvers to the boundary Nevanlinna-Pick interpolation with degree constraint available in the literature, they do not scale to the numerical difficulty or size required, and are based around simplifying assumptions that do not hold for synthesis.

In this thesis I propose two different numerical continuation solvers for synthesis. The methods have lower computational complexity than other guaranteed solvers by using fast linear Fourier transforms to perform polynomial operations. Moreover, the convergence speed of the algorithms is improved by approaching the solution in a novel trajectory. However, the solvers only offer a quicker convergence rate than the heuristic methods for very difficult problems, and may not be effective for real-time applications, as the high-demands in error precision required means a very high computational cost. This seems to be a limitation of the theory, as the high computational cost is a consequence of the large number of iterations required even when the cost of each iteration is low.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Perhaps the defining feature of the modern world is its increasing reliance on the Internet. Ever since its invention, every aspect of society has grown ever more dependent on fast and reliable communication networks. It was all made possible thanks to optical fibers, the backbone of the world's information systems. But, demand continues to grow and the growth does not seem to be ending anytime soon: more and more devices now include a connection to the Internet as a key feature — making them part of what has come to be called the Internet of things — and many more will in the future. And as demand for communication continues to grow, so does the pressure for capacity placed on our current networks. We may reach a point when existing communication infrastructure might no longer be enough [1].

Optical fibers are the only known medium that can satisfy the world's demands for fast and efficient data transfers [1]. But they are a non-linear transmission medium and conventional fiber optic communication assumes that the communication channel is linear and treat non-linearities as noise, which works well at low energy levels, but fails at high power when the non-linearities are excited. This presents a problem because capacity increases with power [1], and hints at a limit — incorrectly but commonly known as the *non-linear Shannon limit* — on the achievable information rate [2], although nobody has proven its existence [1]. Yet, even assuming that the non-linear Shannon limit does not exist, non-linearities are not equivalent to noise and thus conventional fiber-optic communication strategies are not optimal.

Understanding how signal and noise propagate through optical fibers is fundamental for designing new communication schemes that take non-linearities into account. Luckily, it has long been known that the *non-linear Schrödinger equation* describes how electro-magnetic waves travel through ideal non-linear media [3]. Furthermore, it is also known that the *non-linear Fourier transform* offers a solution to the non-linear Schrödinger equation [4].

The non-linear Fourier transform — also known as *scattering transform* [4] — may be considered an extension of Fourier analysis to non-linear problems. However, it only works for certain partial differential equations and is equation specific in a nontrivial way [5]. Nevertheless, non-linear transmission schemes based on the non-linear Fourier transform have recently received interest [6]. The transmission scheme uses the *non-linear Fourier spectrum* to encode the information; then the *inverse non-linear Fourier transform* changes the spectrum to the signal to be sent; and, finally, the *forward non-linear Fourier transform* recovers the spectrum after it has traveled through the optical fiber. The basic insight behind this scheme is that the non-linear Fourier spectrum evolves trivially along an ideal fiber [1].

Finding fast and efficient algorithms for the forward and inverse transform is essential for real-time applications, where computing power is limited [7]. For this purpose, Wahls et al. [8][7] proposed an algorithm for a fast inverse non-linear Fourier transform based around a discretization of the non-linear Schrödinger equation [9]. However, one of the steps in the algorithm — known as the *synthesis* step — only had heuristic solutions so exact methods with theoretical guarantees were still missing. Progress towards a guaranteed solution began in [8], where the problem is defined as an *analytic interpolation* with additional constraints to guarantee that the next step in the fast inverse non-linear Fourier transform algorithm is well-posed. It turns out that this interpolation closely resembles a *Nevanlinna-Pick interpolation with degree constraint*, a problem known to have a guaranteed solution [10]. Indeed, synthesis is a limit case of Nevanlinna-Pick interpolation with degree constraint on the boundary of the region of analyticity.

Although there already are solvers to the boundary Nevanlinna-Pick interpolation with degree constraint in the literature — by Baratchart et al. [11] and Nagamune et al. [12] — they do not scale to the numerical difficulty or size required, and are based around simplifying assumptions that do not hold for synthesis. So in this thesis I extend existing Nevanlinna-Pick interpolation with degree constraint methods to find a guaranteed solution for synthesis.

Outline and Summary of Contributions

The thesis consist of three main parts:

Background

This chapter introduces the general knowledge required to understand the work and the methods in this thesis. It starts with an introduction to polynomials and their numerical representations, an important topic for efficient algorithms. After that, there is an explanation on analytic interpolation, with an emphasis on boundary and Nevanlinna-Pick interpolation with degree constraint. This is followed by an introduction to the fast non-linear Fourier transform developed by Wahls et al. [8] giving a context for the problem this thesis solves. Finally, I formalize the main problem — synthesis using analytic interpolation — and show that it is equivalent to a finite system of non-linear equations.

Numerical Solvers

In this chapter I extend a guaranteed numerical continuation algorithm for boundary analytic interpolation by Baratchart et al. to the synthesis interpolation problem. Moreover, I generalize another continuation algorithm by Nagamune et al. to non-self-congugated datasets. The synthesis interpolation problem can be considered a continuous and smooth mapping from a polynomial to the interpolation values, which is a composition of two functions. The solution is given by the inversion of the mapping, and the algorithms differ in the order of composition. Furthermore, I show that the closed-form partial solution of the synthesis interpolation problem that motivate the heuristics are good initial approximations for the numerical continuation solvers. The new algorithms are used for different cases of synthesis, defocusing and focusing. They can be extended to the other case, but are specially attuned to theirs. Furthermore, I create efficient implementations of the algorithms — faster and able to handle much larger and numerically difficult interpolation datasets — by using fast Fourier transforms.

Results

In this chapter I compare the performance of different heuristics for the closed-form initial approximations. Further, I benchmark three different trajectories for the defocusing case, and one for the focusing case. Interestingly, for numerically difficult problems using Baratchart's approach, the best initial guess is assuming all the iterpolation values are zero. This inital approximation is both closest to the solution in the complex plane and provides the most numerically stable approach to the solution.

I also provide a comparison between the new algorithms and the existing heurisitc solvers. The defocusing solver improves the solution when compared to iterative synthesis only for very difficult problems. Furthermore, it is not possible to achieve arbitrary errors with finite numerical accuracy, as there is a tradeoff between the two possible errors, interpolation and magnitude. In the focusing case, the new solver improves on direct synthesis, but the rate of improvement quickly levels off.

Chapter 2

Background

This chapter introduces the general knowledge required to understand the work and the methods in this thesis. It starts with an introduction to polynomials and the *coefficient* and *point-value* numerical representations, an important topic for efficient algorithms. After that, there is an explanation on *analytic interpolation*, with an emphasis on *Nevanlinna-Pick interpolation with degree constraint*, a technique to compute bounded analytic polynomial functions with low McMillan degree. This is followed by an introduction to the fast non-linear Fourier transform developed by Wahls et al. [8] which gives the motivation for this work, as one of the steps in the algorithm — known as *synthesis* — lacked a guaranteed solution. Finally, I make the relationship between the topics explicit and formalize the main problem: synthesis using analytic interpolation.

2-1 Complex polynomials

Complex polynomials of n *degree* are expressions of the form

$$p(z) := \sum_{k=0}^{n} p_k z^k,$$

where p and z are complex variables. They can be numerically represented in several ways but there two representations that are important from a computational point of view: the *coefficient* and *point-value* representations. These are equivalent, in the sense that every polynomial has a unique point-value and a unique coefficient representation [13].

The *coefficient representation* is the vector (defined by bold lower-case letters) of $n+1$ complex polynomial coefficients

$$\mathbf{p} := \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} \in \mathbb{C}^{n+1}.$$

The *point-value representation* is the set of complex $n+1$ point-value pairs

$$\{\mathbf{z}, \mathbf{p}(\mathbf{z})\}, \quad \mathbf{z} := \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n+1} \end{bmatrix} \in \mathbb{C}^{n+1} \quad \text{and} \quad \mathbf{p}(\mathbf{z}) := \begin{bmatrix} p(z_1) \\ p(z_2) \\ \vdots \\ p(z_{n+1}) \end{bmatrix} \in \mathbb{C}^{n+1},$$

where $\mathbf{z}$ is a vector of distinct points. To keep the notation simple, the point-value representation will be indicated only by the vector of polynomials values $\mathbf{p}(\mathbf{z})$.

Table 2-1: Complexity of naive implementation of polynomial operations of order n [13]

	Sum	Multiplication	Evaluation
Coefficient	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Point-value	$\mathcal{O}(n)$	$\mathcal{O}(2n)$	$\mathcal{O}(n^2)$

Calculating the point-value $\mathbf{p}(\mathbf{z})$ from the coefficient representation $\mathbf{p}$ is a *polynomial evaluation* on $n+1$ points — an operation with a naive implementation of complexity $\mathcal{O}(n^2)$ [13]. However, by choosing the evaluation points as

$$z_k := e^{-i2\pi \frac{k}{n+1}} \quad \text{for } k = 0, 1, \ldots, n, \tag{2-1}$$

the operation is the same as a *fast Fourier transform* (FFT):

$$\boxed{\mathbf{p}} \xrightarrow{\text{FFT}} \boxed{\mathbf{p}(\mathbf{z})}$$

which reduces the complexity to $\mathcal{O}(n \log n)$ [13].

The inverse operation is *polynomial interpolation.* If the *interpolation nodes* $\mathbf{z}$ are given by Eq. (2-1), then computing $\mathbf{p}$ from $\mathbf{p}(\mathbf{z})$ can be done with the *inverse fast Fourier transform* (FFT^{-1}):

$$\boxed{\mathbf{p}(\mathbf{z})} \xrightarrow{\text{FFT}^{-1}} \boxed{\mathbf{p}}$$

also an operation of complexity $\mathcal{O}(n \log n)$ [13].

These are useful because the computational complexity of the naive implementation of common polynomial operations, summarized on Table 2-1, is different for each representation. Notice that every common operation can be computed with linear complexity in one of the representations. An efficient algorithm would switch to that representation to perform the operation reducing the total computational complexity to $\mathcal{O}(n \log n + n) = \mathcal{O}(n \log n)$ for the transformation and the operation in the new representation.

Another important concept for problems with complex variables is *Laurent polynomials*

$$P(z) := \sum_{k=-n}^{n} P_k z^k,$$

a generalization of polynomials to include positive and negative powers and share the same numerical represention s. Their importance to this thesis arises because the square absolute value of a polynomial

$$|p(z)|^2 = p(z)\overline{p(z)} \quad \text{for } |z| = 1,$$

is a Laurent polynomial. This will prove useful later.

2-2 Analytic Interpolation

A function is analytic — also known as *holomorphic* — over some domain if it can be locally represented as a power series. This is equivalent to the function being infinitely differentiable over the whole domain [14]. Analytic interpolation is the construction of an analytic function $f(\cdot)$ — known as an *interpolant* — from a given set of m point-pairs $\{\mathbf{z}, \mathbf{w}\}$. This means finding a function that at the *interpolation nodes*

$$\mathbf{z} := \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \in \mathbb{C}^m,$$

has an *interpolation value* of

$$\mathbf{w} := \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \in \mathbb{C}^m.$$

In other words, a function that satisfies $\mathbf{f}(\mathbf{z}) = \mathbf{w}$, or a function whose graph passes through all the pairs.

Many problems in engineering can be formulated as analytic interpolations. Examples include robust control [15][16][17], circuit and system theory [18], and system identification [19]. In engineering the interpolant $f(\cdot)$ often represents a transfer function [20]. In a discrete-time setting $f(\cdot)$ is the quotient of two complex polynomials

$$f(z) = \frac{b(z)}{a(z)},$$

and system properties are related to analyticity: a *causal* and *stable* system must be *proper* — the degree of $b(z)$ is less or equal to the degree of $a(z)$ — and all the roots of $a(z)$ must be inside the open unit disk

$$a(z) \neq 0 \quad \text{for } |z| \geq 1,$$

or, equivalently, the transfer function $f(z)$ must be be analytic outside the closed unit disk [21].

2-2-1 Nevanlinna-Pick Interpolation with Degree Constraint

The classic solution to analytic interpolation is known as Nevanlinna-Pick interpolation in honor of R. Nevanlinna and G. Pick, who independently solved the problem in the early 20th century [22][23]. A major drawback of the classic theory is that it results in interpolants of large *McMillan degree* (the largest between the degree of the numerator or denominator) [20], which is related to the complexity of the applications — for instance, the degree of the controller or filter in a control system. However, the Nevanlinna-Pick interpolation with degree constraint theory was recently developed to compute analytic interpolants with bounded degree, mainly in the context of robust control [10][20][24].

In this new theory, the interpolation problem is formulated as a constrained convex optimization whose solution satisfies

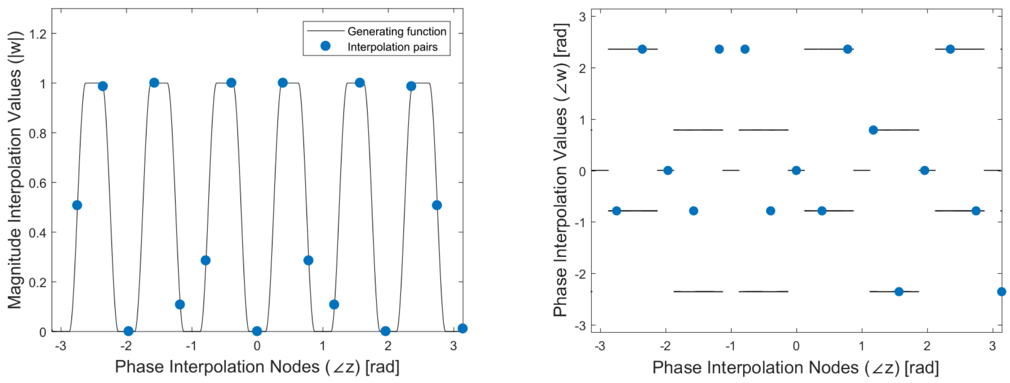
$$|a(z)|^2 - |b(z)|^2 = |c(z)|^2 \quad \text{for } |z| = 1. \tag{2-2}$$

where $b(z)$ is the interpolant's numerator, $a(z)$ is its denominator, and $c(z)$ is a parameterizing polynomial [10].

Nevanlinna-Pick interpolation with degree constraint requires the interpolation nodes to belong to the open unit disk $|z| < 1$, since the gradient of the cost function is infinite at the unit circle $|z| = 1$ [24]. This is problematic for some applications because they naturally lead to interpolation nodes at the boundary of analyticity. For example, the frequency response of a discrete-time dynamic system is the evaluation of its transfer function at $|z| = 1$, so frequency-domain identification is equivalent to analytic interpolation at the boundary, as shown in Figure 2-1. Other examples include loop shaping [24][20], impedance matching [11], and the fast inverse non-linear Fourier transform [8][7].

2-3 The Fast Non-linear Fourier Transform

The non-linear Fourier transform may be considered an extension of Fourier analysis to non-linear problems. However, unlike its linear counterpart, it only works for certain partial differential equations and is equation specific in a nontrivial way [5]. But despite this limitation, the non-linear Fourier transform is useful for many applications [3]. For instance, non-linear transmission schemes based on the non-linear Fourier transform have recently received interest [6]. The basic insight behind them is that the non-linear Fourier spectrum of a signal evolves linearly and trivially along an ideal optical fiber, so encoding information in the non-linear Fourier spectrum eliminates the disturbances caused by the non-linearity of an ideal fiber [1] (other non-linear effects can be reduced to a certain extend [25]). The transmission scheme uses the non-linear Fourier spectrum to encode the information; then the inverse non-linear Fourier transform (NFT^{-1}) changes the spectrum to the signal to be sent;



(a) Magnitude of the frequency response **(b)** Phase of the frequency response

Figure 2-1: Example of discrete-time identification as an interpolation problem, where the nodes **z** are in the unit circle $|z| = 1$ and the frequency response of the system are its values **w**.

and, finally, the forward non-linear Fourier transform (NFT) recovers the spectrum after it has traveled through the optical fiber:

$$\boxed{\text{Encoder}} \rightarrow \overbrace{\boxed{\text{NFT}^{-1}} \rightarrow \boxed{\text{Channel}} \rightarrow \boxed{\text{NFT}}}^{\text{Linearized channel}} \rightarrow \boxed{\text{Decoder}}$$

Fast and efficient algorithms for the forward and inverse transform are essential for real-time applications, where computing power is limited. For this purpose, Wahls et al. [8][7] propose an algorithm for a fast inverse non-linear Fourier transform. Analogous to its linear counterpart, the fast inverse non-linear Fourier transform is an operation on a discrete-time signal. This reflects the realities of digital communication better than a continuous-time system — in digital systems only samples of signal are known — and makes it amenable to computation.

2-3-1 The Non-linear Schrödinger Equation

The fundamental physical action in fiber-optic communication systems is the propagation of a signal in the form of light, an electro-magnetic wave, through optical fibers, a non-linear medium. The signal $q(t, x) \in \mathbb{C}$ evolves in *1+1 dimensions*: a spatial dimension, the location along the fiber $x \geq 0$, and time $t \in \mathbb{R}$ [26]. It has long been known that the non-linear Schrödinger equation [3]

$$i\frac{\partial q(t,x)}{\partial x} + \frac{\partial^2 q(t,x)}{\partial t^2} + 2\gamma|q(t,x)|^2 q(t,x) = 0, \quad \gamma = \pm 1, \tag{2-3}$$

where γ defines the equation as *focusing* ($\gamma = +1$) or *defocusing* ($\gamma = -1$), is an accurate ideal model of how electro-magnetic waves travel through non-linear media. It is an ideal model because it only describes the continuous interaction between fiber dispersion and non-linearity [27], but other non-linear effects can be taken into account to a certain extend [25].

In digital communication, only samples of the signal are known

$$q[k](x) := q(k\epsilon, x) \quad \text{for } k = 0, 1, \dots,$$

and the non-linear Schrödinger equation has to be discretized. There are several possible discretizations of Eq. (2-3), but the Ablowitz-Ladik discretization [9]

$$i\frac{\partial q[k]}{\partial x} + q[k+1] - 2q[k] + q[k-1] + \gamma|q[k]|^2\Big(q[k+1] + q[k-1]\Big) = 0, \quad \gamma = \pm 1, \tag{2-4}$$

has the advantage that it is solvable by a discrete-time version of the non-linear Fourier transform [8].

2-3-2 The Forward Discrete-time Non-linear Fourier Transform

To compute the spatial evolution of a signal evolving as Eq. (2-3), fix a location $x = 0$ and use the forward transform to map a known envelope at the fixed location $q(t, 0)$ to the non-linear Fourier spectrum — also called *scattering data* [3]:

$$\boxed{q(t,0)} \xrightarrow{\text{NFT}} \boxed{\{\hat{q}(z,0), \{\xi_k, \tilde{q}_k(z)\}\}}$$

First, assume the fixed signal $q(t, 0)$ vanishes outside of some interval

$$q(t, 0) = 0 \quad \text{for all } t \notin [t_1, t_2].$$

Then, sampling $q(t, 0)$ on a grid of $m + 1$ points

$$t_k := t_1 + k\epsilon, \quad \epsilon := \frac{t_2 - t_1}{m} \quad \text{for } k = 0, 1, \dots, m,$$

results in the normalized semi-discrete signal — discrete in time, continuous in space —

$$q[k](0) := q\left(\frac{t_n + t_{n+1}}{2}, 0\right).$$

The discrete-time non-linear Fourier transform maps the known fixed semi-discrete signal $q[k](0)$ from a function of the *sampling instant* k to a function of a *spectral variable* z, by using it as a variable in the associated eigenvalue problem [7]

$$\begin{bmatrix} a_k(z) \\ b_k(z) \end{bmatrix} = \frac{1}{\sqrt{1 + \gamma|q[k](0)|^2}} \begin{bmatrix} 1 & q[k](0)z^{-1} \\ -\gamma q[k](0) & z^{-1} \end{bmatrix} \begin{bmatrix} a_{k-1}(z) \\ b_{k-1}(z) \end{bmatrix}, \tag{2-5}$$

with initial conditions

$$\begin{bmatrix} a_0(z) \\ b_0(z) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Eq. (2-5) is a normalized version of the Ablowitz-Ladik discretization of the associated eigenvalue problem for the continuous-time non-linear Schrödinger equation [7].

Iterating Eq. (2-5) m times — until the signal vanishes — results in the discrete-time non-linear Fourier spectrum, which consist of [8]:

- The *continuous spectrum*

$$\hat{q}(z) = \frac{b_m(z)}{a_m(z)}, \quad |z| = 1,$$

- The *discrete spectrum*

$$\{\xi_k, \tilde{q}_k(z)\} \quad \text{for } k = 1, 2, \ldots, n,$$

  where the *eigenvalues* ξ_k are solutions to

$$a_m(\xi_k) = 0 \quad \text{for } |\xi_k| > 1,$$

  and the *norming constants* are

$$\tilde{q}_k = \frac{b_m(\xi_k)}{a'_m(\xi_k)}, \quad a'_m(z) = \frac{\mathrm{d}a_m(z)}{\mathrm{d}z},$$

2-3-3 Spatial Evolution

The spatial evolution of the non-linear Fourier spectrum maps the spectrum of the initial conditions to the spectrum at an arbitrary location:

$$\boxed{\{\hat{q}(z,0), \{\xi_k, \tilde{q}_k(z)\}\}} \xrightarrow[\text{via Eq. (2-4)}]{\text{Spatial evolution}} \boxed{\{\hat{q}(z,x), \{\xi_k, \tilde{q}_k(z)\}\}}$$

The evolution of the continuous spectrum is trivial to solve and given by [3]

$$\begin{aligned} a_m(z,x) &= a_m(z,0), \\ b_m(z,x) &= e^{-4iz^2x} b_m(z,0). \end{aligned}$$

Recall that we are interested in signals with empty discrete spectrum so the evolution of the eigenvalues and norming constants is outside the scope of this work, although it is also trivial [3].

2-3-4 The Fast Inverse Non-linear Fourier Transform

The fast inverse non-linear Fourier transform by Wahls et al. [8][7] only considers signals with empty discrete spectrum, that means that the signal $q[k](x)$ can be fully recovered from the continuous spectrum $\hat{q}(z, x)$. The algorithm has two steps, *synthesis* and *fast layer peeling.*

Synthesis starts with samples of the continuous spectrum on a frequency grid and reconstruct the wave functions of the discrete spectrum $b_m(z)$ and $a_m(z)$ using analytic interpolation. Synthesis is the focus of this thesis, and is explained in more detail in Section 2-4.

The fast non-linear Fourier transform uses a complicated fast layer peeling step, which is outside of the scope of this thesis [8][7]. Conventional layer peeling, on the other hand is simple to understand. It uses $b_m(z)$ and $a_m(z)$ to recover the sampled signal $q[k](x)$ from an inversion of Eq. (2-5)

$$\begin{bmatrix} a_{k-1}(z) \\ b_{k-1}(z) \end{bmatrix} = \frac{1}{\sqrt{1+\gamma|q[k]|^2}} \begin{bmatrix} 1 & -q[k] \\ \gamma\bar{q}[k]z & z \end{bmatrix} \begin{bmatrix} a_k(z) \\ b_k(z) \end{bmatrix}. \tag{2-6}$$

Since $b_k(z)$ and $a_k(z)$ are polynomials of order $k-1$, the first line of Eq. (2-6) is

$$\sqrt{1+\gamma|q[k]|^2} \sum_{j=0}^{k-2} a_{k-1,j} z^j = \sum_{j=0}^{k-1} a_{k,j} z^j - q[k] \sum_{j=0}^{k-1} b_{k,j} z^j.$$

Comparing the coefficients of z^{k-1} results in $a_{k,k-1} = q[k] b_{k,k-1}$, so

$$q[k](x) = \frac{b_{k,k-1}}{a_{k,k-1}}. \tag{2-7}$$

The algorithm recovers the complete signal $q[m](x), q[m-1](x), \ldots, q[0](x)$ by iterating Eq. (2-7) and Eq. (2-6). The polynomials $b_m(z)$ and $a_m(z)$ found by synthesis are used as initial conditions [8].

2-4 Synthesis as an Analytic Interpolation Problem

Synthesis is the recovery of the wave functions of the discrete non-linear Fourier spectrum

$$b_m(z) := \sum_{n=0}^{m-1} b_{m,n} z^{-n} \quad \text{and} \quad a_m(z) := \sum_{n=0}^{m-1} a_{m,n} z^{-n},$$

from a set of interpolation node-value pairs $\{\mathbf{z}, \mathbf{w}\}$. The interpolation nodes

$$\mathbf{z} := \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \in \mathbb{C}^m, \quad z_k := e^{i2\pi\frac{k-1}{m}} \quad \text{for } k = 1, 2, \ldots, m, \tag{2-8}$$

are a frequency grid of m points in the unit circle $|z| = 1$, and the interpolation values

$$\mathbf{w} := \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \in \mathbb{C}^m, \quad w_k = \hat{q}(z_k) \quad \text{for } k = 1, 2, \ldots, m, \tag{2-9}$$

are set to match the continuous Fourier spectrum on the frequency grid. However, $b_m(z)$ and $a_m(z)$ are the inputs to layer peeling and must be chosen such that it is well-posed [7] and the complete signal is recoverable. This issue is called *realizability* and adds three additional constraints

1. $|a_m(z)|^2 + \gamma |b_m(z)|^2 = 1$,
2. $a_m(z) \neq 0$ for all $|z| > 1$,
3. $a_{m,0} \in \mathbb{R}$ and $a_{m,0} \geq 0$.

Every $a_m(z)$ and $b_m(z)$ constructed by the forward non-linear Fourier transform satisfies these. Moreover if they are satisfied, there exist samples that lead to them.

Notice that defocusing synthesis is a boundary Nevanlinna-Pick interpolation with degree constraint since the solution parameterized by $c(z) = 1$ in Eq. (2-2) meets Condition 1; Conditions 2 is guaranteed by analytic interpolation; and Condition 3 is trivial to solve and only requires multiplying $a_m(z)$ by a unimodal constant.

The main problem in this thesis can be then formalized as:

Problem 1 (Synthesis via Analytic Interpolation). *Given an ordered set of m complex distinct equidistant interpolation nodes on the unit circle*

$$\boldsymbol{z} := \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \in \mathbb{C}^m, \quad z_k := e^{i2\pi \frac{k-1}{m}} \quad \textit{for } k = 1, 2, \ldots, m,$$

and an ordered set of m complex interpolation values

$$\boldsymbol{w} := \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \in \mathbb{C}^m.$$

Find two complex polynomials of order $m - 1$

$$b(z) := \sum_{n=0}^{m-1} b_n z^{-n} \quad \textit{and} \quad a(z) := \sum_{n=0}^{m-1} a_n z^{-n},$$

satisfying the following conditions:

1. ***Interpolation***

$$b(z_k) = w_k a(z_k) \quad \textit{for } k = 1, 2, \ldots, m. \tag{2-10}$$

2. ***Magnitude***

$$|a(z)|^2 + \gamma |b(z)|^2 = 1. \tag{2-11}$$

3. ***Stability***

$$a(z) \neq 0 \quad \textit{for } |z| > 1. \tag{2-12}$$

Chapter 3

Solvers

The defocusing and focusing synthesis interpolation have a few important differences, so it is worth exploring them separately. Historically, problems similar to analytic interpolation in the defocusing case have been more researched [17], hence they are better understood and their solutions are more sophisticated. Indeed, synthesis in the defocusing case is a limit case of Nevanlinna-Pick interpolation with degree constraint proposed by Byrnes et al. [10], where the interpolation nodes $\mathbf{z}$ lie on $|z| < 1$ instead of $|z| = 1$.

The general idea behind the solver is known as *numerical continuation* [28] and is explained in more detail in Section 3-1. I present two variations of the technique, one for each case. Each have special implications and are specially suited for their case, even though both are variations of the same basic idea. They still find a solution in the other case, but they are impractical and run into numerical issues.

Numerical continuation is not guaranteed to converge for all problems, however it changes the scope of the problem from one global to many local ones. This change of scope offers a mathematical guarantee for analytic interpolation because the local problems have guaranteed solutions even when directly solving the global problem does not [28].

3-1 The Defocusing Case

The sign of γ places some additional restrictions on the input data and solution. For defocusing interpolation, Eq. (2-11) implies that $|a(z)| > |b(z)|$, and hence $|\mathbf{w}| < 1$. Another implication of the defocusing case is that the problem requires higher *numerical precision* (the number of bits used by the digital representations) as $|\mathbf{w}| \to 1$, since Eq. (2-10) means that $b(z) \to a(z)$ as $|\mathbf{w}| \to 1$, but as $b(z)$ and $a(z)$ grow increasingly equal, only $b(z) \to \infty$ and $a(z) \to \infty + 1$ will satisfy both Eq. (2-11) and Eq. (2-10).

There are two stages to the solution of Problem 1: initial explicit approximations that satisfy some of the realizability conditions fully and some only partially, and iterative solvers that use the explicit approximation as a starting points and iterate towards a better solution.

3-1-1 Explicit Approximations

I propose three closed-form approximations. The first one, the *zero approximation*, is given by

$$b(z) = 0 \quad \text{and} \quad a(z) = z^{m-1}, \tag{3-1}$$

which satisfies the magnitude and the stability conditions — Eq. (2-11) and Eq. (2-12), respectively — and do not rely on the interpolation input data $\{\mathbf{z}, \mathbf{w}\}$. Although simple, this approximations has numerical benefits for the iterative solver, explained in Chapter 4.

There are other explicit approximations that more closely satisfy the realizability conditions. Using $\{\mathbf{z}, \mathbf{w}\}$ to solve simplified versions of Problem 1 results in two sets: the *defocusing explicit approximations* (Algorithm 1, pg. 25), which fully satisfy Eq. (2-11), but only match the magnitude of the interpolation values $|\mathbf{w}|$; and the *focusing explicit approximations* (Algorithm 4, pg. 27), which satisfy the interpolation condition of Eq. (2-10) but only guarantee Eq. (2-11) at $\mathbf{z}$. Each set corresponds naturally to one of the solvers.

Computing the Magnitude of the Polynomials

The magnitude of the point-value representation of the polynomials, $|\mathbf{b}(\mathbf{z})|$ and $|\mathbf{a}(\mathbf{z})|$, can be computed explicitly [7], and is the first step in finding the explicit approximations. To do so, we rewrite the interpolation condition from Eq. (2-10) in polar coordinates,

$$|b(z_k)| = |w_k||a(z_k)| \quad \text{and} \quad \angle b(z_k) = \angle w_k + \angle a(z_k) \quad \text{for } k = 1, 2, \ldots, m. \tag{3-2}$$

Then, by considering the magnitude condition of Eq. (2-11) only on the interpolation nodes

$$|a(z_k)|^2 + \gamma |b(z_k)|^2 = 1 \quad \text{for } k = 1, 2, \ldots, m, \tag{3-3}$$

we can compute the magnitude of the point-value representation of the polynomials by replacing Eq. (3-2) into Eq. (3-3) to find

$$|b(z_k)| = \frac{|w_k|}{\sqrt{1 + \gamma |w_k|^2}} \quad \text{and} \quad |a(z_k)| = \frac{1}{\sqrt{1 + \gamma |w_k|^2}} \quad \text{for } k = 1, 2, \ldots, m. \tag{3-4}$$

Nothing is known about the phases $\angle \mathbf{b}(\mathbf{z})$ yet, so the point-value representation of the polynomial is not complete. However, any two polynomial satisfying Eq. (3-4) will guarantee that Eq. (2-10) holds up to $|\mathbf{w}|$, and that Eq. (2-11) is satisfied at $\mathbf{z}$. With this we can define the two sets of approximations.

The Defocusing Explicit Approximations

The first set are closed-form approximations satisfying the magnitude condition of Eq. (2-11) and the stability condition of Eq. (2-12), but the interpolation only up to the magnitude

$$\left|\frac{b(z_k)}{a(z_k)}\right| = |w_k| \quad \text{for } k = 1, 2, \ldots, m. \tag{3-5}$$

For any given $b(z)$, there is a unique $a(z)$ that satisfies Eq. (3-4) and Eq. (2-12) [11], which can be found via *spectral factorization* [29]

$$a(z) = \sigma(1 - \gamma|b(z)|^2) \quad \text{for } |z| = 1, \tag{3-6}$$

an operation that maps a polynomial (or Laurent polynomial) of degree $2n$ positive on the unit circle, to a *minimum phase polynomial* (a polynomial with all its roots inside the unit disk) and another with all its roots outside the unit disk — spectral factorization is the inverse operation of multiplying a minimum phase polynomial with its conjugate [29]. So, starting with a $\mathbf{b}(\mathbf{z})$ with magnitude given by Eq. (3-4) and computing $a(z)$ from Eq. (3-6) results in polynomials that satisfy Eq. (2-11), Eq. (2-12), and Eq. (3-5). The whole process is summarized by Algorithm 1 (pg. 27).

The resulting polynomials, $b(z)$ and $a(z)$, form a set parameterized by the phases of the point-value representation $\angle\mathbf{b}(\mathbf{z})$. They can be considered a generalization of the solution to Problem 8. in [8] and to the initial conditions of iterative synthesis in [7] because the solution to those problems — found by setting $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{w}$ — is included in the set of defocusing explicit approximations.

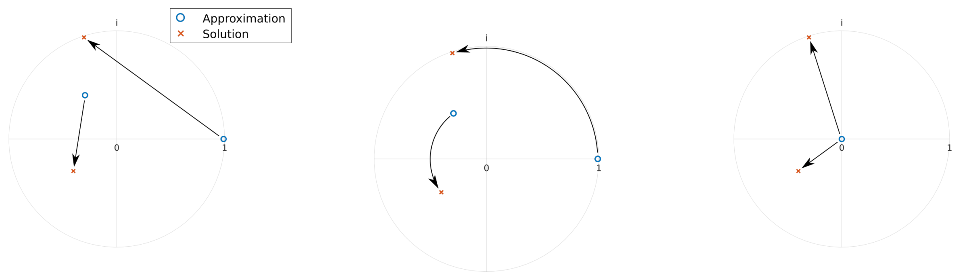
3-1-2 Defocusing Iterative Solver

Numerical continuation is an approach to solving a system of non-linear equations by turning them into an ordinary differential equation. I will only provide an overview of numerical continuation as it relates to the specifics of the solvers, for an in-depth and more rigorous discussion of the topic see Allgower and Georg [28], and Sydel [30].

The defocusing iterative solver (Algorithm 3, pg. 26) is similar to an algorithm by Baratchart et al. [11]. It starts by transforming the problem into a dynamic system by making the polynomials into functions of a parameter t,

$$b(z,t) = \sum_{n=0}^{m-1} b_n(t) z^{-n} \quad \text{and} \quad a(z,t) = \sum_{n=0}^{m-1} a_n(t) z^{-n}.$$

Then, it defines $a(z,t)$ as the unique polynomial satisfying Eq. (2-11) and Eq. (2-12) for a given $b(z,t)$ by using Eq. (3-6). This leaves $b(z,t)$ the only independent variable and the problem becomes finding a $b(z,t)$ that satisfies Eq. (2-10). To achieve this, the algorithm travels a path from some known initial guess at $t = 0$ to the solution at $t = 1$. This path is defined as a known curve $\mathbf{w}(t)$ between initial conditions $\mathbf{w}(0)$ that are known or easy to



(a) Linear **(b)** Circular **(c)** Radial

Figure 3-1: Graphical representation of the continuation trajectories for the defocusing solver.

compute, and the solution $\mathbf{w}(1) = \mathbf{w}$. The original non-linear problem becomes the *homotopy* (a continuous function that deforms a function into another)

$$H\Big(b(z_k, t)\Big) := b(z_k, t) - a(z_k, t)w_k(t) = 0 \quad \text{for all } t \quad \text{and } k = 1, 2, \ldots, m. \tag{3-7}$$

The initial conditions are given by any arbitrary $b(z, 0)$, but using the approximations of Section 3-1-1 is preferable because they are closer to the solution or have benefits for numerical implementations — the specific are discussed in Chapter 4.

Next, find a path $\mathbf{w}(t)$ between the initial conditions $\mathbf{w}(0)$ and the desired answer $\mathbf{w}(1) = \mathbf{w}$. Any continuous smooth function with a known $\mathbf{w}(0)$ and $\mathbf{w}(1) = \mathbf{w}$ can be used as a continuation trajectory, but the problem leads naturally to three paths, represented graphically in Figure 3-1. These trajectories are:

- A simple *linear path*, commonly used in continuation [28],

$$w_k(t) := w_k(0) + t(w_k(1) - w_k(0)) \quad \text{for } k = 1, 2, \ldots, m, \tag{3-8}$$

$$\frac{\mathrm{d}\mathbf{w}(t)}{\mathrm{d}t} = \mathbf{w}(1) - \mathbf{w}(1). \tag{3-9}$$

- A *circular path*, taking advantage of the magnitude information of (3-4),

$$w_k(t) := |w_k(1)|e^{i(\angle w_k(0) + t(\angle w_k(1) - \angle w_k(0)))} \quad \text{for } k = 1, 2, \ldots, m, \tag{3-10}$$

$$\frac{\mathrm{d}w_k(t)}{\mathrm{d}t} = w_k(t)i(\angle w_k(1) - \angle w_k(0)) \quad \text{for } k = 1, 2, \ldots, m, \tag{3-11}$$

- A *radial path*, following a linear path starting with the zero approximation,

$$w_k(t) = tw_k(1), \quad k = 1, 2, \ldots, m, \tag{3-12}$$

$$\frac{\mathrm{d}\mathbf{w}(t)}{\mathrm{d}t} = \mathbf{w}(1). \tag{3-13}$$

It is important to clarify that it is not possible to solve the continuation problem by only approximating the phases since Eq. (2-11) gives no information about them, so the problem is under-determined without the magnitude information. But the circular path does something similar by approaching the phases linearly and interpolating the magnitudes exactly at every point along the path.

In the following, I will only explain in the procedure for the linear path of Eq. (3-8), but using the circular or radial paths is straight-forward.

Turning the Problem into an Ordinary Differential Equation

We can compute the ordinary differential equation from the derivative of Eq. (3-7)

$$\frac{\mathrm{d}b(z_k,t)}{\mathrm{d}t} = \frac{\mathrm{d}a(z_k,t)}{\mathrm{d}t} w_k(t) + a(z_k,t)\frac{\mathrm{d}w_k(t)}{\mathrm{d}t} \quad \text{for } k = 1,2,\dots,m. \tag{3-14}$$

To compute $\mathrm{d}a(z_k,t)/\mathrm{d}t$ we use the derivative of Eq. (2-11)

$$\overline{a(z,t)}\frac{\mathrm{d}a(z,t)}{\mathrm{d}t} + a(z,t)\frac{\mathrm{d}\overline{a(z,t)}}{\mathrm{d}t} = \overline{b(z,t)}\frac{\mathrm{d}b(z,t)}{\mathrm{d}t} + b(z,t)\frac{\mathrm{d}\overline{b(z,t)}}{\mathrm{d}t}, \tag{3-15}$$

and solve for

$$\frac{\mathrm{d}a(z,t)}{\mathrm{d}t} = \frac{\overline{b(z,t)}}{\overline{a(z,t)}}\frac{\mathrm{d}b(z,t)}{\mathrm{d}t} \tag{3-16}$$

Lastly, substituting Eq. (3-16) and Eq. (3-13) in Eq. (3-14) results in

$$\begin{aligned}\frac{\mathrm{d}b(z_k,t)}{\mathrm{d}t} &= w_k(t)\frac{\overline{b(z_k,t)}}{\overline{a(z_k,t)}}\frac{\mathrm{d}b(z_k,t)}{\mathrm{d}t} + \frac{\mathrm{d}w_k(t)}{\mathrm{d}t}a(z_k,t),\\ &= \frac{(w_k(1)-w_k(0))a(z_k,t)\overline{a(z_k,t)}}{\overline{a(z_k,t)} - w_k(t)\overline{b(z_k,t)}} \quad \text{for } k = 1,2,\dots,m.\end{aligned} \tag{3-17}$$

Traversing the Trajectory

To go from the initial known solution $\mathbf{b}(\mathbf{z},0)$ to the answer $\mathbf{b}(\mathbf{z},1)$ we use a *predictor-corrector* iterative method. The method consist of two steps per iteration. The *predictor step* predicts the direction of the trajectory based on its derivative, and follows the line tangent to the curve by a specified step-length α, resulting in a new vector

$$\mathbf{b}(\mathbf{z},t+\alpha) = \mathbf{b}(\mathbf{z},t) + \alpha\frac{\mathrm{d}\mathbf{b}(\mathbf{z},t)}{\mathrm{d}t}.$$

However, this vector may deviate from the homotopy, that is, it may not satisfy Eq. (3-7). So the *corrector step* pulls the vector back to the path.

The corrector step is a local version of Problem 1, with interpolation values $\mathbf{w}(t)$ instead of $\mathbf{w}$. Locality is important because Newton-Rhapson's root finding algorithm

$$\begin{aligned} b(z_k, t) &= b(z_k, t) - \frac{H\Big(b(z_k, t)\Big)}{H'\Big(b(z_k, t)\Big)}, \\ H' := \frac{\partial H}{\partial b} &= 1 - w(z_k, t)\frac{\overline{b(z_k, t)}}{\overline{a(z_k, t)}} \quad \text{for } k = 1, 2, \ldots, m. \end{aligned} \tag{3-18}$$

is guaranteed to find the roots of Eq. (3-7) for a sufficiently small α [12][11]. This means that finding the roots of Eq. (3-7) is a convex problem around the trajectory. And since the roots of Eq. (3-7) at $t = 1$ are the solution to Problem 1, the continuation method is guaranteed to find a solution.

Additionally, the step-length α should be adapted at each iteration for efficient computation. If the step-length is too large the correction step might need many iterations to push the polynomials back into the path, or it might not converge at all. But if α is too small the algorithm will need many prediction steps, which would take too long. Furthermore, since the answer is given at $t = 1$, the step-length adaptation also stops the algorithm from overstepping the solution. To achieve this, Algorithm 3 (pg. 25) uses the linear deformation

$$\alpha := \alpha_0(1 - t) + \alpha_1 t,$$

because the problem gets numerically harder as $\mathbf{w}(t) \to \mathbf{w}$ so smaller steps are required at the end.

3-2 The Focusing Case

In focusing interpolation Eq. (2-11) implies that $|a(z)| \leq 1$ and $|b(z)| \leq 1$, but the interpolation values $\mathbf{w}$ are not bounded. This means that the focusing case requires greater numerical precision as $\mathbf{w} \to \infty$, since Eq. (2-10) implies that $|a(z)| \to 0$ as $\mathbf{w} \to \infty$.

3-2-1 Explicit Approximations

Just as in the defocusing case, the solver uses a closed-form approximation as a starting point for the solver. But, as far as I can tell, there is no initial point that makes the solver behave better numerically, so the starting point is the one that most closely approximates the solution, which is $\angle\mathbf{a}(\mathbf{z}) = 0$ — the comparison is available in Chapter 4.

Explicit Focusing Approximations

The starting polynomials belong to the explicit focusing approximations (Algorithm 4 pg. 27) introduced in Section 3-1-1, which are closed-form solutions satisfying the interpolation of Eq. (2-10) perfectly, but only satisfy the magnitude condition of Eq. (2-11) at the interpolation nodes $\mathbf{z}$. Similarly to the explicit defocusing approximations (Algorithm 1 pg. 25), they start with a polynomial $a(z)$ whose point-value magnitude is given by Eq. (3-4) and use spectral factorization to compute an $a(z)$ that meets the stability condition of Eq. (2-12). Then, define $b(z)$ as the polynomial satisfying Eq. (2-10) by establishing the map

$$b(z_k) = w_k a(z_k) \quad \text{for } k = 1, 2, \ldots, m. \tag{3-19}$$

The resulting $a(z)$ and $b(z)$ are parameterized by the phase of the point-value representation $\angle\mathbf{a}(\mathbf{z})$ before the spectral factorization. Hence, the explicit focusing approximations set is a generalization of *direct synthesis* in [7], in the same way that the defocusing approximations (Algorithm 1 pg. 25) generalizes the initial guess of iterative synthesis.

Comparing different heuristics shows that $\angle\mathbf{a}(\mathbf{z}) = 0$ is the closest closed-form approximation, so it will be the starting point of the solver. This comparison is presented in Section 4-3-1.

3-2-2 Iterative Focusing Solver

Numerical continuation is used to iterate towards a better solution by a process resembling Nagamune's Nevanlinna-Pick interpolation with degree constraint solver [12], and summarized by Algorithm 5 (pg. 28). The basic idea arises from the fact that the solutions to the Nevanlinna-Pick interpolation with degree constraint problem are parameterized by a stable polynomial of $m-1$ degree [10]

$$c(z) := \sum_{n=0}^{m-1} c_n z^{-n},$$

which satisfies

$$|a(z)|^2 + |b(z)|^2 = |c(z)|^2.$$

So, the algorithm iterates towards a solution from an initial polynomial $c(z, 0)$, computed with the explicit focusing approximation, following a linear path

$$c(z, t) := c(z, 0) + t\Big(1 - c(z, 0)\Big).$$

$$\frac{\partial c(z, t)}{\partial t} = 1 - c(z, 0). \tag{3-20}$$

The iteration is analogous to the defocusing interpolation (Algorithm 3, pg. 26), but now the algorithm seeks to numerically solve Eq. (2-11), an infinite-dimensional problem. However, Eq. (2-11) can be reformulated as a finite-dimensional system of equations — the proof is in Chapter B — defined on an over-sampled vector

$$\mathbf{x} := \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{2m-1} \end{bmatrix} \in \mathbb{C}^{2m-1},$$

since it involves Laurent polynomials of order $2m-1$: $|a(z)|^2$, $|b(z)|^2$, and $|c(z)|^2$.

The homotopy is defined as

$$H\Big(a(x_k,t)\Big) := |a(x_k,t)|^2 + |b(x_k,t)|^2 - |c(x_k,t)|^2 = 0 \quad \text{for } k = 1,2,\ldots,2m-1. \tag{3-21}$$

The predictor step is

$$\mathbf{a}(\mathbf{z}, t+\alpha) = \mathbf{a}(\mathbf{z},t) + \alpha \frac{\mathrm{d}\mathbf{a}(\mathbf{z},t)}{\mathrm{d}t}, \tag{3-22}$$

with $\mathrm{d}\mathbf{a}(\mathbf{z},t)/\mathrm{d}t$ given by replacing Eq. (3-19) into the derivative of Eq. (3-21),

$$\overline{a(x_k,t)}\frac{\mathrm{d}a(x_k,t)}{\mathrm{d}t} = \overline{c(x_k,t)}\frac{\mathrm{d}c(x_k,t)}{\mathrm{d}t} - \overline{b(x_k,t)}\frac{\mathrm{d}b(x_k,t)}{\mathrm{d}t} \quad \text{for } k = 1,2,\ldots,2m-1,$$

to find

$$\frac{\mathrm{d}a(x_k,t)}{\mathrm{d}t} = \frac{\overline{c(x_k,t)}}{\overline{a(x_k,t)} + w_k\overline{b(x_k,t)}}\frac{\mathrm{d}c(x_k,t)}{\mathrm{d}t} \quad \text{for } k = 1,2,\ldots,2m-1.$$

The corrector is a Newton-Rhapson root finding algorithm on Eq. (3-21) for a constant $c(x_k,t)$

$$\begin{aligned} a(x_k,t) &= a(x_k,t) - \frac{H\Big(a(x_k,t)\Big)}{H'\Big(a(x_k,t)\Big)}, \\ H' := \frac{\partial H}{\partial a} &= \overline{a(x_k,t)} + \overline{b(x_k,t)}\frac{\mathrm{d}b(x_k,t)}{\mathrm{d}a} \quad \text{for } k = 1,2,\ldots,2m-1. \end{aligned} \tag{3-23}$$

where $H\Big(a(x_k,t)\Big)$ is given by Eq. (3-21) and

$$\frac{\mathrm{d}b(x_k,t)}{\mathrm{d}a} = w_k = \frac{b(x_k,t)}{a(x_k,t)}, \quad k = 1,2,\ldots,2m-1, \tag{3-24}$$

is a vector of over-sampled interpolation values.

3-3 Discussion and Proof of Convergence

Algorithm 3 (pg. 26) and Algorithm 5 (pg. 28) are related to several methods in the literature. Problem 1 in the defocusing case is an extension of the boundary Nevanlinna-Pick interpolation with degree constraint for impedance matching of Baratchart et al. [11] from interpolation nodes in the real line to the unit circle. Baratchart's impedance matching is itself a generalization of Nevanlinna-Pick interpolation with degree constraint theory by Byrnes et al. [10] to include nodes in the boundary of analyticity. The equivalence between impedance matching and synthesis in the defocusing case is proven by the conformal transformation [31]

$$x = \frac{i(1-z)}{z+1},$$

where x belongs in the upper half-plane and $|z| = 1$.

There are several important results for synthesis in [11]. The First result is that spectral factorization is continuous and smooth for Laurent polynomials with no roots on the unit circle. The second one is given by Theorem 1, which states that the map

$$\psi\Big(b(z)\Big) = \begin{bmatrix} \frac{b(z_1)}{a(z_1)} \\ \vdots \\ \frac{b(z_m)}{a(z_m)} \end{bmatrix} \tag{3-25}$$

where $b(z)$ satisfies Eq. (2-10) and Eq. (2-11) is continuous and smooth — and hence differentiable — with a continuous and smooth inverse. The map in Eq. (3-25) is the composition of a non-linear map

$$b(z) \to \{b(z), a(z)\}, \tag{3-26}$$

where $a(z)$ is the unique polynomial satisfying Eq. (2-11) for any given $b(z)$, followed by the evaluation map

$$\{b(z), a(z)\} \to \frac{b(z_k)}{a(z_k)} \quad \text{for } k = 1, 2, \dots, m. \tag{3-27}$$

The solution to Problem 1 is given by

$$b(z) = \psi^{-1}(\mathbf{w}), \tag{3-28}$$

which is the inversion of Eq. (3-25) evaluated at the desired interpolation values. The third important result is given by Proposition 9, which states that for every $\epsilon > 0$ there is a path $b(z,t)$ such that

$$\sup_{t\in[0,1]} \Big\| \psi\Big(\mathbf{b}(\mathbf{z},t)\Big) - \mathbf{w}(t) \Big\| \leq \epsilon.$$

Under these conditions, the continuation methods proposed in this chapter converge. In general, numerical continuation is guaranteed to converge under the following assumptions [28]:

1. There is a smooth curve $b(z,t) \in H^{-1}(0)$ with $b(z,0) \in b(z,t)$. This condition is equivalent to Eq. (3-30) being smooth.
2. If the curve exist, it reaches the target $t = 1$ in a finite length. This is assured by the uniqueness and continuity of Eq. (3-30).

Notice that these are extendable to the focusing case of Algorithm 3 (pg. 26) on the condition that $|b(z)| < 1$ for $|z| = 1$ — a sufficient condition for the existence of $a(z)$ and $b(z)$ satisfying Eq. (2-11) in the focusing case. Baratchart method is then guaranteed to find a solution for Problem 1. My contribution is creating Algorithm 3 (pg. 26), a fast and efficient algorithm able to handle significantly larger and more difficult problems.

Algorithm 3 (pg. 26) is also related to Sander's et al. iterative synthesis [7] as both algorithms follow the same basic idea: define $a(z)$ as the polynomial satisfying Eq. (2-11) for any given $b(z)$, and solve Eq. (2-10) numerically. Algorithm 3 (pg. 26) with a circular path is particularly close, as both algorithms only correct the phase $\angle\mathbf{b}(\mathbf{z})$ and not the magnitude. The difference is that iterative synthesis makes $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{a}(\mathbf{z}) + \angle\mathbf{w}$ after each iteration — that is, it makes $\angle\mathbf{b}(\mathbf{z})$ whatever solves Eq. (2-10) for the current $a(z)$ — while Algorithm 3 (pg. 26) approaches the solution from a predefined path.

On the other hand, Algorithm 5 (pg. 28) is related to Nagamune's et al. solver for Nevanlinna-Pick interpolation with degree constraint [24] (available at `https://people.kth.se/~ryozo/software.html`). The solver reverts the order of composition for the homotopy and solves the problem in terms of $a(z)$. The mapping becomes

$$\hat{\psi}\Big(a(z)\Big) := a(z) \to \{b(z), a(z) | \frac{b(z_k)}{a(z_k)} = w_k \quad \text{for } k = 1, 2, \dots, m\} \to c(z). \tag{3-29}$$

where $c(z)$ is a positive real Laurent polynomial. The solution to Problem 1 is

$$a(z) = \hat{\psi}^{-1}(1). \tag{3-30}$$

Changing the order of composition does not change continuity or smoothness so Algorithm 5 (pg. 28) also fulfills the convergence conditions for numerical continuation. On a similar note, Algorithm 5 (pg. 28) is a guaranteed way to iterate direct synthesis [7] to a better solution.

3-4 Algorithms

Algorithm 1: Defocusing Explicit Approximation

```
  Input: $\mathbf{z}$, $\mathbf{w}$, $\angle\mathbf{b}(\mathbf{z})$.
  Output: $b(z)$, $a(z)$
1 Find the magnitudes $|\mathbf{b}(\mathbf{z})|$
      $$|b(z_k)| = \frac{|w_k|}{\sqrt{1-|w_k|^2}}, \quad \text{for } k = 1, 2, \ldots, m.$$
2 Compute the point-value representation
      $$b(z_k) := |b(z_k)| \exp^{i\angle b(z_k)}, \quad \text{for } k = 1, 2, \ldots, m.$$
3 Transform $\mathbf{b}(\mathbf{z})$ into the coefficient representation by using the inverse Fourier
    transform ($\mathrm{FFT}^{-1}$)
      $$b(z) = \mathrm{FFT}^{-1}(\mathbf{b}(\mathbf{z}), m).$$
4 Calculate the oversampled point-value representation by using fast Fourier transform
    (FFT)
      $$\mathbf{b}(\mathbf{x}) = \mathrm{FFT}(b(z), 2m-1).$$
5 Construct the Laurent polynomial
      $$B(x_k) := |b(x_k)|^2, \quad \text{for } k = 1, 2, \ldots, 2m-1.$$
6 Find $\mathbf{a}(\mathbf{z})$ satisfying the magnitude condition of Eq. (2-11) and the stability condition
    of Eq. (2-12) by spectral factorization of $\mathbf{B}(\mathbf{x}) + 1$
      $$\mathbf{a}(\mathbf{z}) = \sigma\Big(\mathbf{B}(\mathbf{x}) + 1\Big).$$
```

Algorithm 2: Iterative Synthesis

```
   Input: $\mathbf{z}$, $\mathbf{w}$, $\mathbf{a}(\mathbf{z})$.
   Output: $b(z)$, $a(z)$
1  repeat
2      Compute $\mathbf{b}(\mathbf{z})$ satisfying the interpolation condition of Eq. (2-10)
           $b(z_k) = a(z_k)w_k \quad \text{for } k = 1, 2, \ldots, m.$
3      Construct the Laurent polynomial
           $B(x_k) := |b(x_k)|^2 \quad \text{for } k = 1, 2, \ldots, 2m-1.$
4      Find $\mathbf{a}(\mathbf{z})$ satisfying the magnitude condition of Eq. (2-11) and the stability
         condition of Eq. (2-12) by spectral factorization of $\mathbf{B}(\mathbf{x}) + 1$
           $\mathbf{a}(\mathbf{z}) = \sigma\Big(\mathbf{B}(\mathbf{x}) + 1\Big).$
5  until until convergence;
```

Algorithm 3: Defocusing Iterative Solver

```
   Input: $\mathbf{z}$, $\mathbf{w}(t)$, $\mathbf{b}(\mathbf{z}, 0)$, $\alpha$.
   Output: $b(z)$, $a(z)$
1  Set the variables to the initial conditions
           $t = 0$
2  Iterate towards a solution while $t <= 1$ do
3      Update the path
           $t = t + \alpha.$
4      Predictor: predict the next point in the curve $b(z, t)$ with Euler method
           $\mathbf{b}(\mathbf{z}, t) = \mathbf{b}(\mathbf{z}, t) + \alpha \frac{\mathrm{d}\mathbf{b}(\mathbf{z}, t)}{\mathrm{d}t}.$
5      Corrector: improve the magnitude error $e_2$ with Newton-Rhapson root finding
         repeat
6              $b(z_k, t) = b(z_k, t) - \frac{H\Big(b(z_k, t)\Big)}{H'\Big(b(z_k, t)\Big)},$
               $H' := \frac{\partial H}{\partial b} = 1 - w(z_k, t)\frac{\overline{b(z_k, t)}}{a(z_k, t)} \quad \text{for } k = 1, 2, \ldots, m.$
7      until until convergence;
8      Update the step-length $\alpha$.
9  end
```

```
Algorithm 4: Focusing Explicit Approximation
  Input: $\mathbf{z}$, $\mathbf{w}$, $\angle\mathbf{a}(\mathbf{z})$.
  Output: $b(z)$, $a(z)$
1 Find the magnitudes $|\mathbf{a}(\mathbf{z})|$
      $|a(z_k)| = \frac{1}{\sqrt{1+|w_k|^2}} \quad \text{for } k = 1, 2, \ldots, m.$
2 Compute the point-value representation
      $a(z_k) := |a(z_k)| \exp^{i\angle a(z_k)} \quad \text{for } k = 1, 2, \ldots, m.$
3 Transform $\mathbf{a}(\mathbf{z})$ into the coefficient representation by using the inverse Fourier
    transform ($\text{FFT}^{-1}$)
      $a(z) = \text{FFT}^{-1}(\mathbf{a}(\mathbf{z}), m).$
4 Calculate the oversampled point-value representation by using fast Fourier transform
    (FFT)
      $\mathbf{a}(\mathbf{x}) = \text{FFT}(b(z), 2m-1).$
5 Construct the Laurent polynomial
      $A(x_k) := |a(x_k)|^2 \quad \text{for } k = 1, 2, \ldots, 2m-1.$
6 Find $\mathbf{a}(\mathbf{z})$ satisfying the stability condition of Eq. (2-12) by spectral factorization of
    $\mathbf{A}(\mathbf{x})$
      $\mathbf{a}(\mathbf{z}) = \sigma\Big(\mathbf{A}(\mathbf{x})\Big).$
7 Compute $\mathbf{b}(\mathbf{z})$ satisfying the interpolation condition of Eq. (2-10)
      $b(z_k) = a(z_k) w_k \quad \text{for } k = 1, 2, \ldots, m.$
```

Algorithm 5: Focusing Iterative Solver

```
  Input: $\mathbf{z}$, $\mathbf{w}$, $\mathbf{a}(\mathbf{z}, 0)$, $\alpha$.
  Output: $b(z)$, $a(z)$
1 Set the variables to the initial conditions
      $$t = 0$$
2 Iterate towards a solution while $t <= 1$ do
3     Update the path
          $$t = t + \alpha.$$
4     Predictor: predict the next point in the curve $a(z, t)$ with Euler method
          $$\mathbf{a}(\mathbf{z}, t) = \mathbf{a}(\mathbf{z}, t) + \alpha \frac{\mathrm{d}\mathbf{a}(\mathbf{z}, t)}{\mathrm{d}t}.$$
5     Corrector: improve the magnitude error $e_2$ with Newton-Rhapson root finding
       repeat
6          $$a(x_k, t) = a(x_k, t) - \frac{H\Big(a(x_k, t)\Big)}{H'\Big(a(x_k, t)\Big)},$$
           $$H' := \frac{\partial H}{\partial a} = \overline{a(x_k, t)} + \overline{b(x_k, t)} \frac{\mathrm{d}b(x_k, t)}{\mathrm{d}a} \quad \text{for } k = 1, 2, \ldots, 2m - 1.$$
7     until *until convergence*;
8     Update the step-length $\alpha$.
9 end
```

Chapter 4

Results and Discussion

In this chapter I analyze the performance of the algorithms with respect to the quantity of interpolation points and the numerical difficulty of the problem, and compare them with existing heuristic algorithms by Walhs and Vishal [7]. Comparing the solvers to conventional guaranteed techniques is not possible for interpolation data similar to a real application of non-linear Fourier transform, since the solvers available in the literature — a Nevanlinna-Pick interpolation with degree constraint by Nagamune and Blomqvist [12], and a boundary interpolation solver by Baratchart et al. [11] — cannot solve the problem. Nevertheless, a comparison between Algorithm 3 (pg. 26) and Nagamune et al. solver is presented for interpolation data suitable for computation by both algorithms. It was not possible to obtain Baratchart's et al. solver, but they confirmed by correspondence that their solver would not be suitable for synthesis.

Interestingly, for the defocusing case the closest explicit approximation is not the best starting point for the iterative solver, because it runs into numerical issues earlier. Instead the best starting point is the zero approximation, since it approaches the solution radially so the local solutions are always as far away from the unit circle as possible — the numerical issues of the defocusing case are caused by the interpolation values being close to the unit circle. The defocusing iterative algorithm satisfies the interpolation to an arbitrary value but there is a trade-off between satisfying Eq. (2-10) and Eq. (2-11). This trade-off is caused by numerical inaccuracies, as Eq. (2-11) should always hold by definition. Furthermore, the comparison to heuristic methods show that only numerically difficult problems benefit from using defocusing iterative solver (Algorithm 3, pg. 26) over iterative synthesis (Algorithm 2, pg. 26), but the advantages are large.

The focusing iterative algorithm (Algorithm 5, pg. 28) starts with the closest explicit approximation since there are no numerical issues in the iterations. Also, its difficult to develop an intuition into a better continuation path — in contrast to the defocusing case — so only a linear trajectory was evaluated. Although it should be possible to obtain an arbitrary magnitude error, the rate of change of the corrector step for numerically difficult problems becomes too small to make the Newton-Rhapson root finding algorithm practical. As a consequence, the error in Eq. (2-11) for the focusing solution remains large, but the solution is still an improvement over direct synthesis (Algorithm 4, pg. 27).

4-1 Test Functions for Performance Analysis

The test functions are the same as the those used by Walhs et al. in [7]. The data was generated by setting the interpolation nodes $\mathbf{z}$ to be m equidistant points in the unit circle given by Eq. (2-8), and the interpolation values $\mathbf{w}$ were set to match samples of the continuous-time continuous spectrum $\hat{q}(\lambda_k)$ on a frequency grid

$$\lambda_k := i\frac{m}{2(t_2 - t_1)}\log z_k,$$
$$w_k := \hat{q}[k] := e^{i\lambda_k(t_2+t_1)}z^{\frac{m+1}{2}}\hat{q}(\lambda_k) \quad \text{for } k = 1, 2, \ldots, m.$$

The algorithms were evaluated with respect to the number of interpolation points m and the difficulty of solving the problem numerically, determined by the *maximum reflectivity*

$$\rho := 1 - \max|\hat{q}[k]|,$$

in the defocusing case, and by the *inverse maximum amplitude*

$$\rho := \frac{1}{\max|\hat{q}[k]|},$$

in the focusing case. A lower ρ means that the problem is numerically harder to solve [7].

4-1-1 Defocusing Test Function: Hyperbolic Secant

The hyperbolic test function will be used for the defocusing case. Its continuous spectrum is [32],

$$\hat{q}\left(\lambda_k\right) = -2^{-2i\mathcal{F}}\mathcal{Q}\frac{\Gamma\Big(d_k\Big)\Gamma\Big(f_{k-}\Big)\Gamma\Big(f_{k+}\Big)}{\Gamma\Big(\overline{d_k}\Big)\Gamma\Big(g_{-}\Big)\Gamma\Big(g_{+}\Big)},$$

where $\Gamma(\cdot)$ is the gamma function with arguments

$$\begin{aligned} d_k &= 0.5 + i\left(\lambda_k\mathcal{L} - \mathcal{F}\right)\right), \\ f_{k\pm} &= 0.5 - i\left(\lambda_k\mathcal{L} \pm \sqrt{\mathcal{F}^2 + \mathcal{Q}^2}\right), \\ g_{\pm} &= 1 - i\left(\mathcal{F} \pm \sqrt{\mathcal{F}^2 + \mathcal{Q}^2}\right), \quad \text{for } k = 1, 2, \ldots, m. \end{aligned}$$

where $\mathcal{F} = 1.5$, $\mathcal{L} = 1/25$, and $\mathcal{Q}$ is related to ρ, with larger values generating more difficult problems — i.e. lower ρ. An example with $m = 4096$ interpolation points and maximum reflectivity $\rho = 0.0776$ is shown in Figure 4-1 and Figure 4-2.

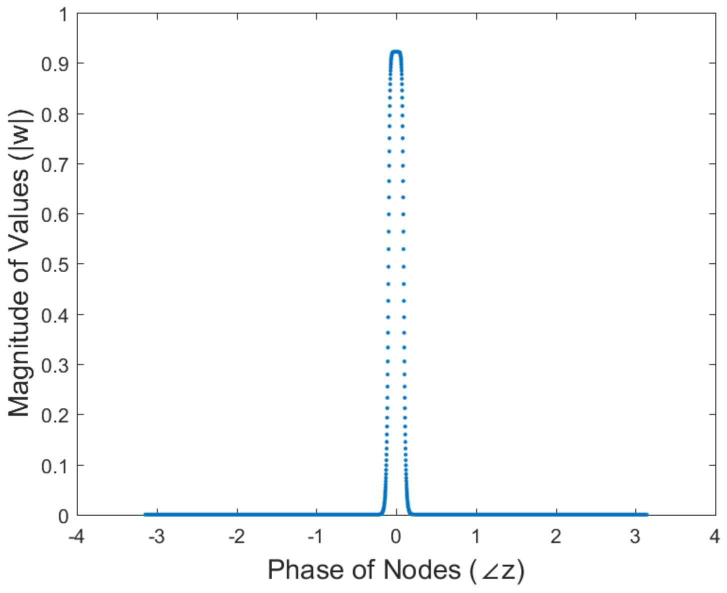


Figure 4-1: Magnitude of interpolation values generated by a hyperbolic secant continuous spectrum with $m = 4096$ points, maximum reflectivity $\rho = 0.0776$, and range from $t_1 = -2$ to $t_2 = 2$.

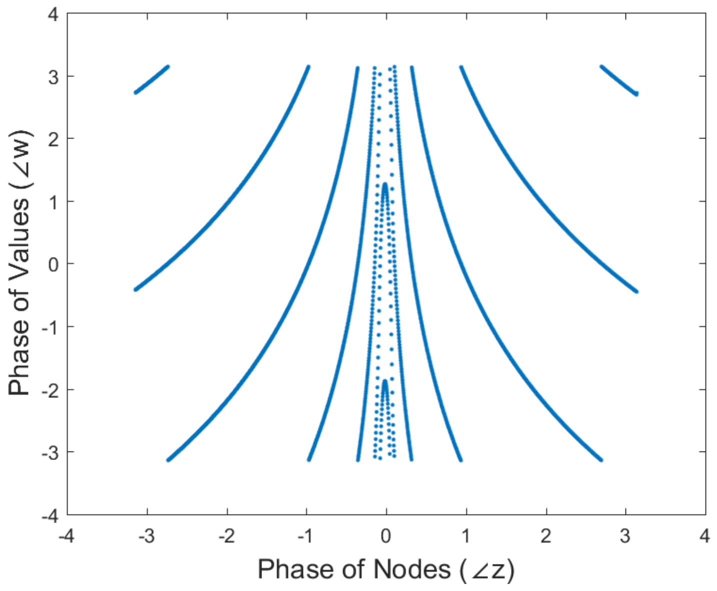


Figure 4-2: Phase of interpolation values generated by a hyperbolic secant continuous spectrum with $m = 4096$ points, maximum reflectivity $\rho = 0.0776$, and range from $t_1 = -2$ to $t_2 = 2$.

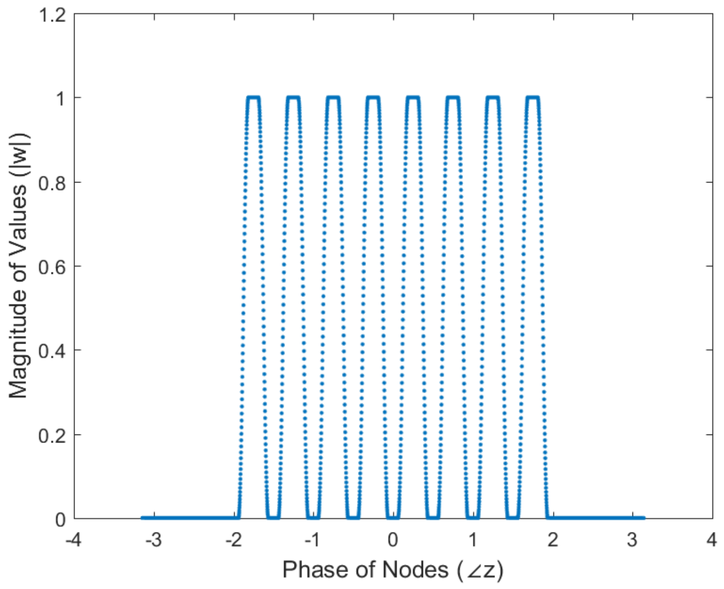


Figure 4-3: Magnitude of interpolation values generated by an eight raised cosines continuous spectrum with $m = 4096$ points, inverted maximum amplitude $\rho = 1$, and range from $t_1 = -512$ to $t_2 = 512$.

4-1-2 Focusing Test Function: Eight Raised Cosines

The eight raised cosines will be used for the focusing case. Its continuous spectrum is [7],

$$\hat{q}\left(\lambda_k\right) = \frac{1}{\rho}\sum_{j=1}^{8} e^{i\theta_j} r\left(\lambda_k - c_j\right),$$

with centers $c_j = j - 4.5$, phases θ_j chosen randomly from $\{\pm 0.75\pi, \pm 0.25\pi\}$, and a normalized raised cosine filter

$$r(\lambda_k) = \begin{cases} 1, & |\lambda_k| \leq \frac{1-\beta}{2W} \\ 0.5 + 0.5\cos\left(\frac{\pi W}{\beta}\left(|\lambda_k| - \frac{1-\beta}{2W}\right)\right), & \frac{1-\beta}{2W} < |\lambda_k| \leq \frac{1+\beta}{2W} \\ 0, & \text{Otherwise} \end{cases}$$

with width $W = 2$ and roll-off factor $\beta = 0.5$. A focusing example with $m = 4096$ interpolation node-value pairs and $\rho = 1$ is shown in Figure 4-3 and Figure 4-4.

4-2 Performance Metrics

The performance of the interpolation condition of Eq. (2-10) is measured by the *interpolation error* [7]

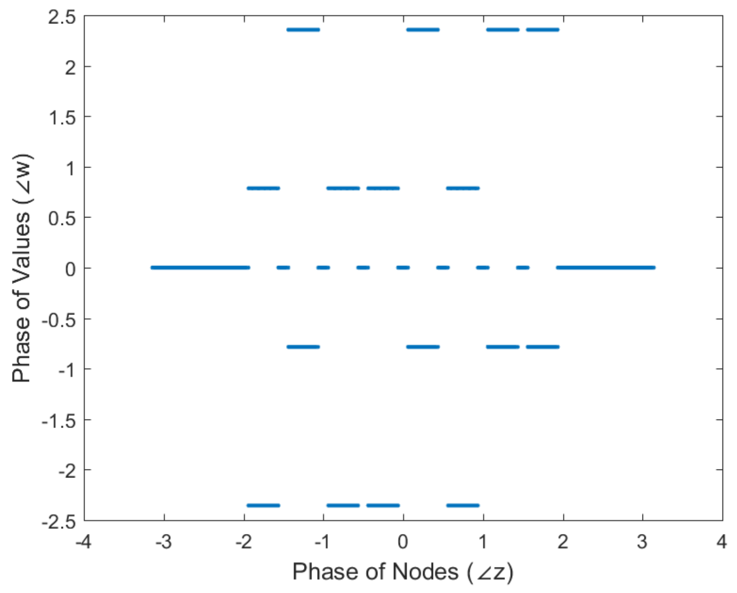


Figure 4-4: Phase of interpolation values generated by an eight raised cosines continuous spectrum with $m = 4096$ points, inverted maximum amplitude $\rho = 1$, and range from $t_1 = -512$ to $t_2 = 512$.

$$e_1 := \frac{\|w_k - b(z_k)/a(z_k)\|}{\|w_k\|} \quad \text{for } k = 1, 2, \ldots, m.$$

The infinite-dimensional magnitude condition of Eq. (2-11) can be reformulated as a finite system of $2m-1$ equations — proof is given in Chapter B — and its satisfaction is measured by the *magnitude error*

$$e_2 := \frac{1}{2m-1}\||a(x_k)|^2 + \gamma|b(x_k)|^2 - 1\| \quad \text{for } k = 1, 2, \ldots, 2m-1.$$

evaluated on an oversampled node vector

$$\mathbf{x} := \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{2m-1} \end{bmatrix} \in \mathbb{C}^{2m-1} \quad \text{for } |x_k| = 1 \quad \text{for all } k,$$

where x_k are distinct points in the unit circle.

4-3 The Defocusing Case

4-3-1 Comparison of Explicit Approximations

Four different heuristics for the phase of the point-value representation $\angle\mathbf{b}(\mathbf{z})$ were evaluated with respect to the number of interpolation node-value pairs m and maximum reflectivity ρ. They are compared for the defocusing hyperbolic secant, with $m = 2^8, \ldots, 2^{20}$ interpolation node-value pairs and $\mathcal{Q} = 0.5, \ldots, 5.5$.

The theoretical magnitude error for these approximations is $e_2 = 0$ since $a(z)$ is computed to satisfy Eq. (2-11). Polynomials that satisfy Eq. (3-5) are computed explicitly so the interpolation error is completely determined by failure of the interpolant to match the phase of the interpolation values $\angle\mathbf{w}$. Assuming the magnitude of this error is uniformly distributed on the range $[0, \pi]$ gives an average phase error of $\pi/2$, which results in an expected interpolation error of $E(e_1) = \sqrt{2}$.

The test cases were:

- Matching the phase of the interpolation values: $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{w}$.
- Zero phase: $\angle\mathbf{b}(\mathbf{z}) = 0$.
- Constant shift in the phase of the interpolation values: $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{w} + \pi/4$.
- Random phase: $\angle\mathbf{b}(\mathbf{z})$ = uniform random variable in $[0, 2\pi]$.

The closest approximation is to make $\angle b(z_k) = \angle w_k$. This is most noticeable in the interpolation error e_1 seen in Figure 4-5, as it has lower e_1 for all m. However, it is only a better approximation for simple problems — functions with large maximum reflectivity ρ — as Figure 4-7, Figure 4-8 and Figure 4-9 show that it performs no better that random in other cases. The magnitude error graphs in Figure 4-13, Figure 4-14, Figure 4-10, and Figure 4-12 also show that the approximation $\angle b(z_k) = \angle w_k$ is better in general, since it has lower e_1 and e_2 for functions with large ρ and similar errors otherwise.

The number of interpolation pairs m has little influence compared to ρ. Specially on the magnitude error e_2, easily seen in Figure 4-13 and Figure 4-14, as e_2 decreases considerably — from over 10^{10} to less than 10^{-10} — for increasing ρ. Yet, e_2 decreases much more slowly for increasing m, as can be seen in Figure 4-10 and Figure 4-12. Furthermore, Figure 4-5 and Figure 4-7 show that increasing m makes e_1 grow very little for simple problems and change randomly for difficult problems

Also, the very large e_2 for problems with very small ρ hints at the importance of numerical accuracy in the performance of the solution. In theory, there should not be any magnitude error — because $a(z)$ is defined as the polynomial satisfying the magnitude condition of Eq. (2-11) — so any magnitude error is caused by an inaccurate computation of $a(z)$.

4-3-2 Comparison of Continuation Trajectories

Numerical issues play a big role in the results. Specifically, the polynomials $b(z)$ and $a(z)$ are increasingly equal as $\mathbf{w} \to 1$, which is equivalent to $\rho \to 0$. At the limit, there is no

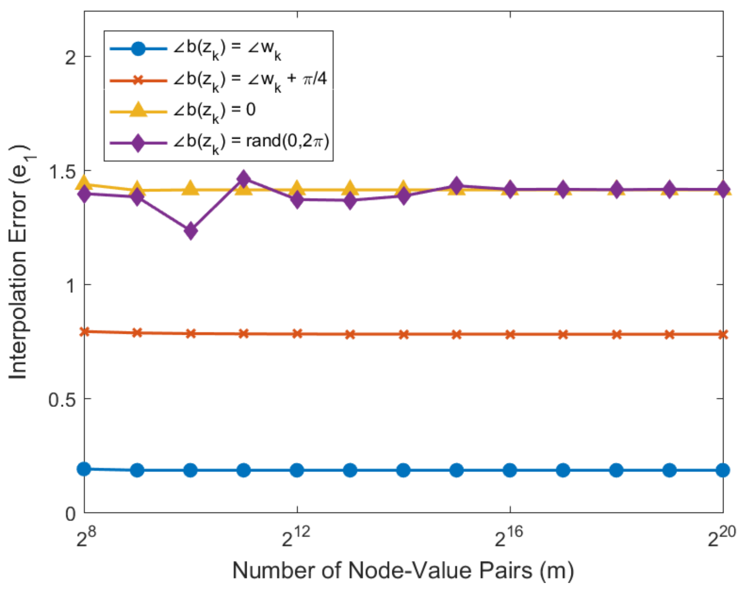


Figure 4-5: Comparison of the interpolation error e_1 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the number of interpolation pairs m using a hyperbolic secant with $\mathcal{Q} = 0.5$ as the generating function. Matching the phase of the values, i.e. $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{w}$, results in lower e_1 for any m. Furthermore, there is no noticeable change in e_1 with respect to m.

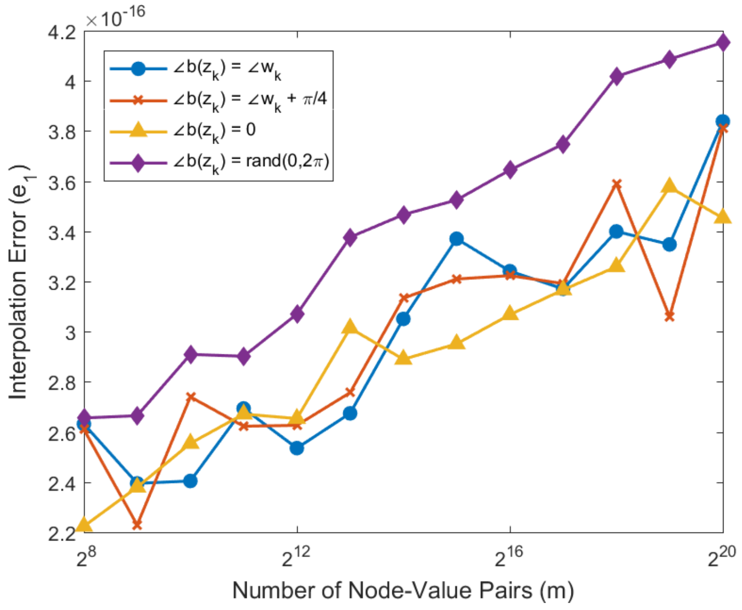


Figure 4-6: Comparison of the interpolation error e_1 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the number of interpolation pairs m using a hyperbolic secant with $\mathcal{Q} = 3.5$ as the generating function. There is no noticeable relationship between $\angle\mathbf{b}(\mathbf{z})$ and e_1.

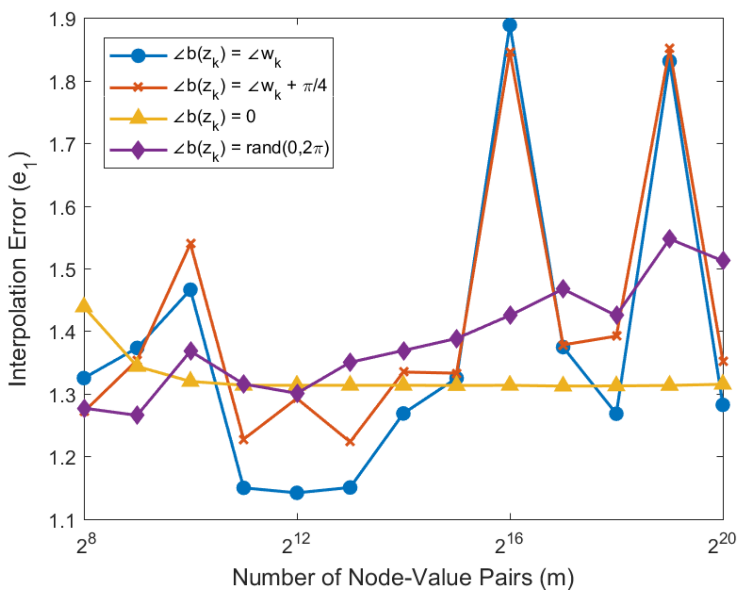


Figure 4-7: Comparison of the interpolation error e_1 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the number of interpolation pairs m using a hyperbolic secant with $\mathcal{Q} = 5.5$ as the generating function. There is no noticeable relationship between e_1 and m as the changes are arbitrary, or very small.

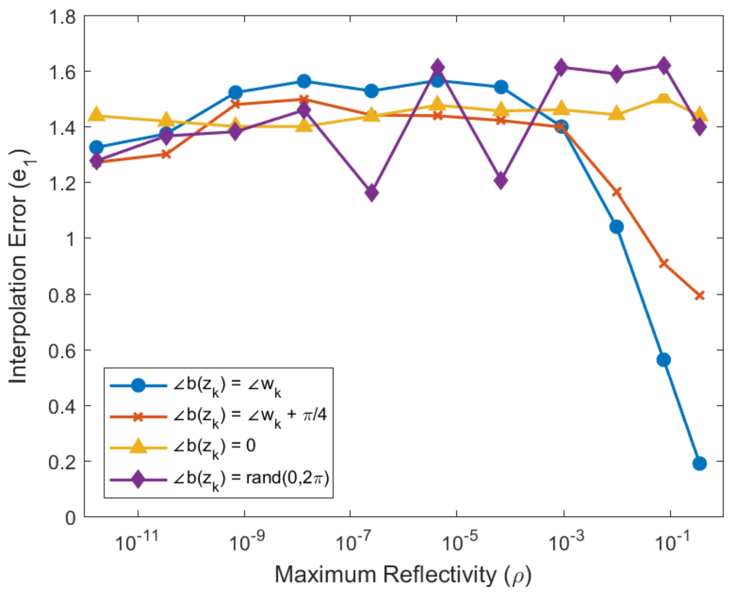


Figure 4-8: Comparison of the interpolation error e_1 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the maximum reflectivity ρ using a hyperbolic secant with $m = 2^8$ interpolation points as the generating function. Matching the phase of the values, i.e. $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{w}$, results in lower e_1 for problem with high ρ, but its performance is no better than random otherwise.

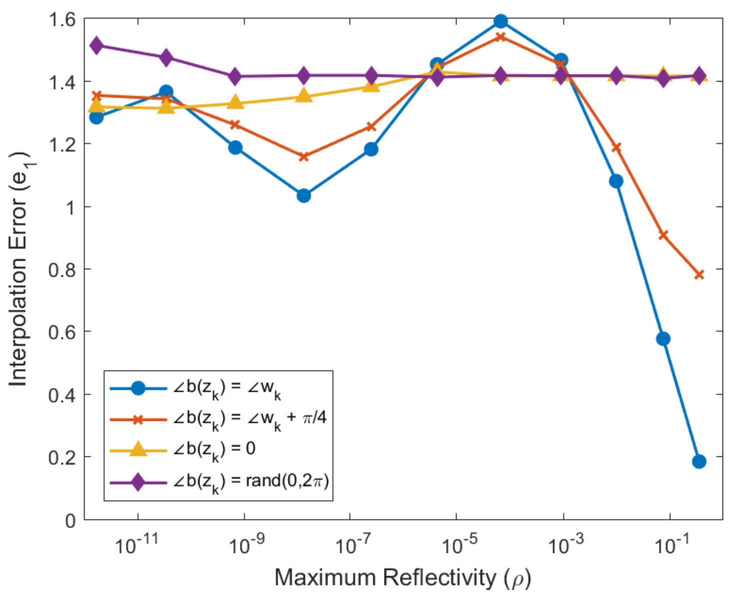


Figure 4-9: Comparison of the interpolation error e_1 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the maximum reflectivity ρ using a hyperbolic secant with $m = 2^{20}$ interpolation points as the generating function. Matching the phase of the values, i.e. $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{w}$, results in lower e_1 for problem with high ρ, but its performance is no better than random otherwise.

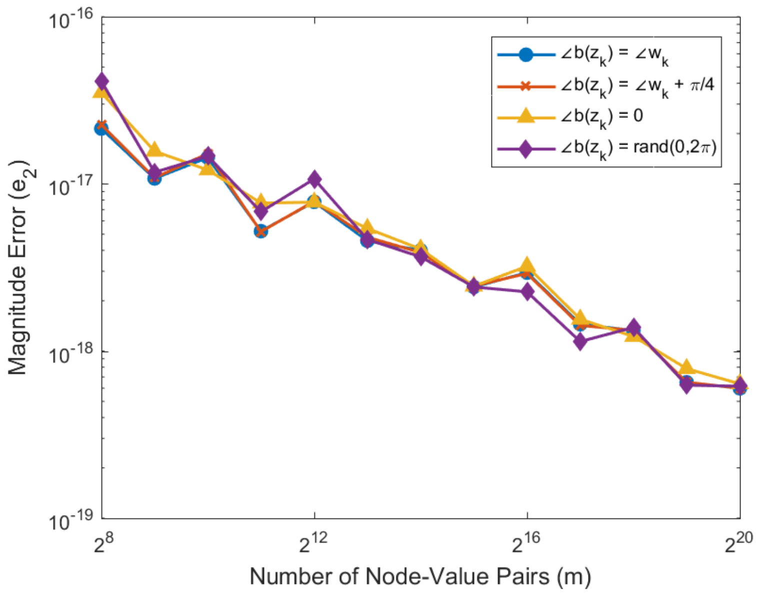


Figure 4-10: Comparison of the magnitude error e_2 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the number of interpolation pairs m using a hyperbolic secant with $\mathcal{Q} = 0.5$ as the generating function. The phase of the point-value representation $\angle\mathbf{b}(\mathbf{z})$ have no effect on e_2, but e_2 is inversely proportional to m.

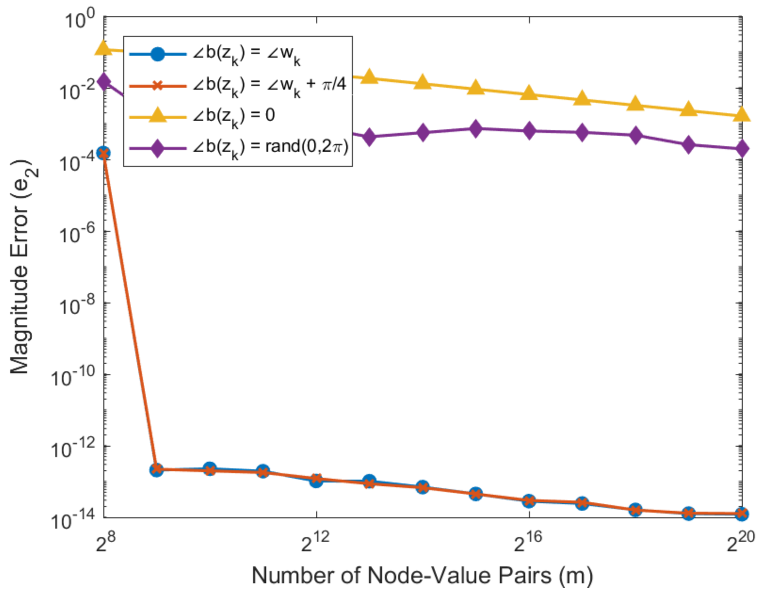


Figure 4-11: Comparison of the magnitude error e_2 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the number of interpolation pairs m using a hyperbolic secant with $\mathcal{Q} = 3.5$ as the generating function. Matching the phase $\angle\mathbf{b}(\mathbf{z}) = \angle\mathbf{w}$ and a constant phase shift result in lower e_2.

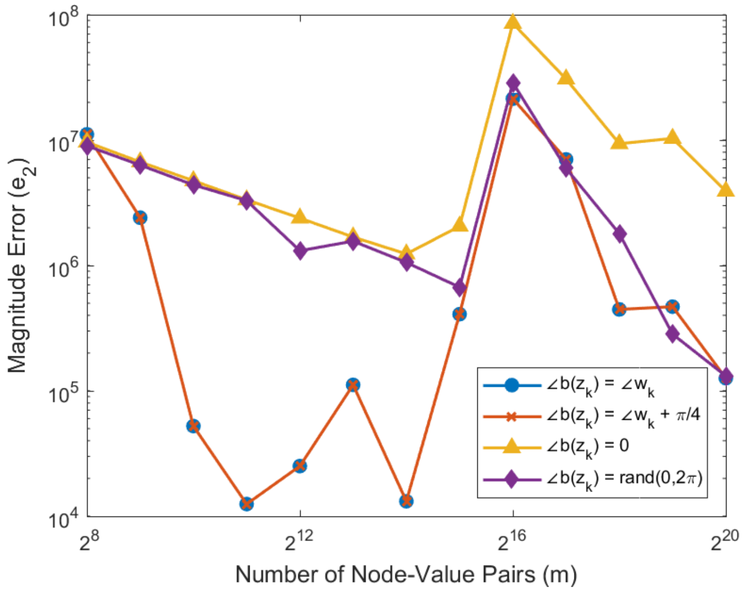


Figure 4-12: Comparison of the magnitude error e_2 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the number of interpolation pairs m using a hyperbolic secant with $\mathcal{Q} = 5.5$ as the generating function. The phase of the point-value representation $\angle\mathbf{b}(\mathbf{z})$ have no effect on e_2, but e_2 is inversely proportional to m, although only slightly.

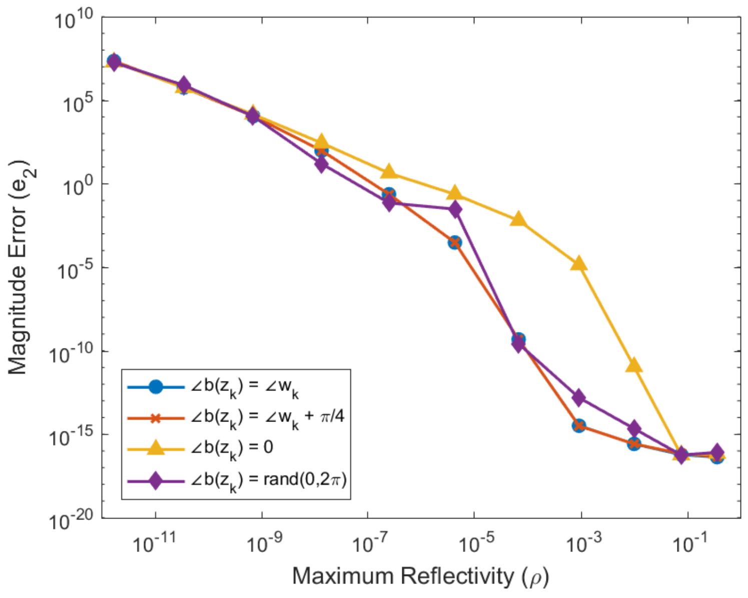


Figure 4-13: Comparison of the magnitude error e_2 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the maximum reflectivity ρ using a hyperbolic secant with $m = 2^8$ interpolation points as the generating function. The magnitude error e_2 increases significantly for decreasing ρ. Furthermore, the phase of the point-value representation of the polynomial $b(z)$ have no effect on e_2.

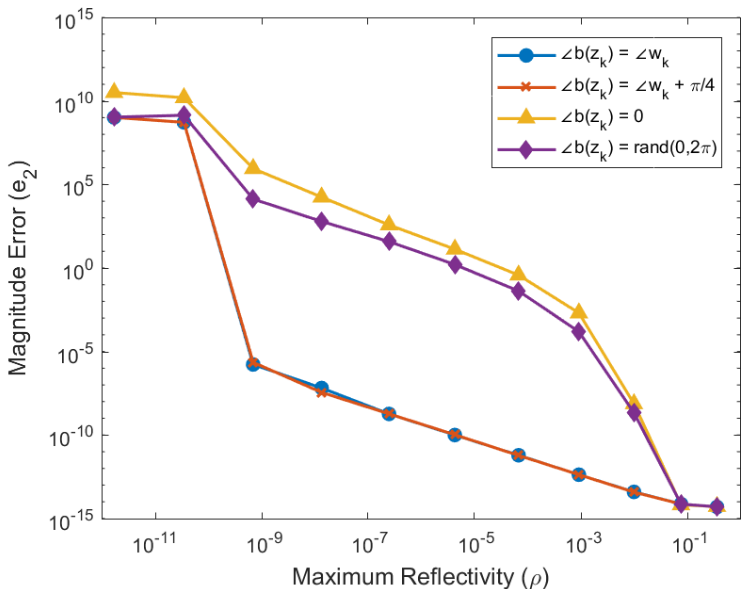


Figure 4-14: Comparison of the magnitude error e_2 of defocusing explicit approximations (Algorithm 1 pg. 25) with respect to the maximum reflectivity ρ using a hyperbolic secant with $m = 2^{20}$ interpolation points as the generating function. The magnitude error e_2 increases significantly for decreasing ρ. Furthermore, matching the phase of the values, i.e. $\angle \mathbf{b}(\mathbf{z}) = \angle \mathbf{w}$, results in lower e_2.

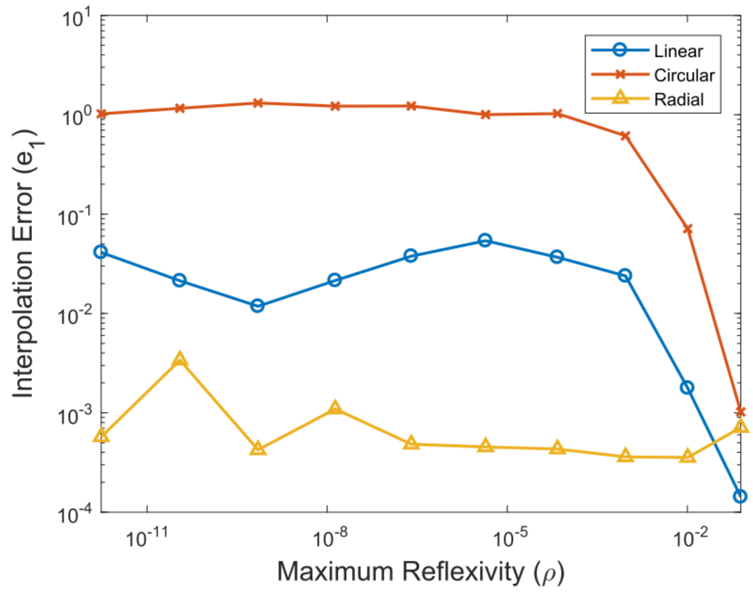


Figure 4-15: Comparison of the interpolation error e_1 of three continuation paths $\mathbf{w}(t)$ with respect to maximum reflectivity ρ for 100 iterations of the defocusing solver (Algorithm 3 pg. 26). The radial path achieves the lowest e_1.

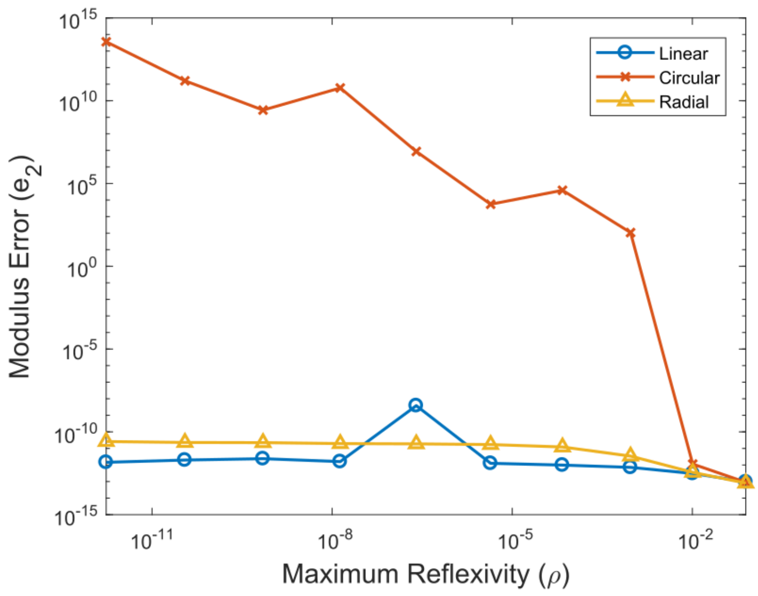


Figure 4-16: Comparison of the magnitude error e_2 of three continuation paths $\mathbf{w}(t)$ with respect to maximum reflectivity ρ. The large magnitude error e_2 makes the circular path inviable, while the radial and linear paths have comparable e_2.

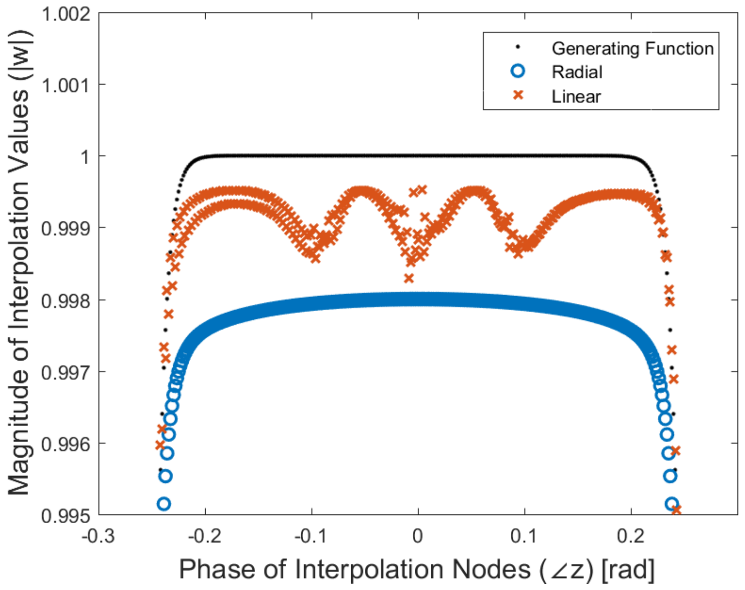


Figure 4-17: Comparison of the magnitude of the frequency response of the generating function and interpolants found by different continuation trajectories with similar interpolation error e_1. The radial path results in an interpolant with the shape of the magnitude frequency response closer to that of the generating function.

solution satisfying both Eq. (2-10) and Eq. (2-11), so the map of Eq. (3-6) grows increasingly inaccurate when $b(z)$ and $a(z)$ satisfy Eq. (3-5).

This is the key problem for the circular path. In theory, we should expect better results since we are making use of the magnitude information retrieved explicitly in Section 3-1-1, and only need to approach the phases. However, the magnitude of some of the local interpolation values remain close to unit circle at every iteration so Eq. (3-6) remains inaccurate, for the reasons explained in the previous paragraph. Furthermore, the predictor step follows a line tangent to the trajectory so the continuation path is not followed perfectly (this is the reason for the corrector step), which risks pushing some $\mathbf{w}(t)$ outside the unit disk making $|\mathbf{a}(\mathbf{z}, t)| > 1$ — an inviable initial state for the corrector step — causing the algorithm to stop working. We can see this happening in Figure 4-16, where approaching the trajectory in a circular path results in a large magnitude error e_2 due to inaccurate computations.

A close visual inspection of Figure 4-17 and Figure 4-18, which shows the final shape of solutions for a linear and radial paths with similar interpolation errors e_1 — 3.2×10^{-3} for radial and 9.7×10^{-3} for linear — reveals that the radial path is closer to the expected magnitude shape in Figure 4-17, but it is farther away from the interpolation values closest to one. However, the radial path interpolates the phase of Figure 4-18 very well so its e_1 is lower. The reason for this difference is that the local interpolation values along the radial path, i.e. $\mathbf{w}(t) = t\mathbf{w}$, are scalar transformations of the desired interpolation values, hence preserving the shape of the generating function; while the the linear path have different rates of change. Furthermore, numerical issues on $|z| = 1$ reinforce why the zero approximation is the best starting point: it is the closet to the solution in the complex plane for difficult problems, with $e_1 = 1$ by definition, and the most numerically stable path is to approach the

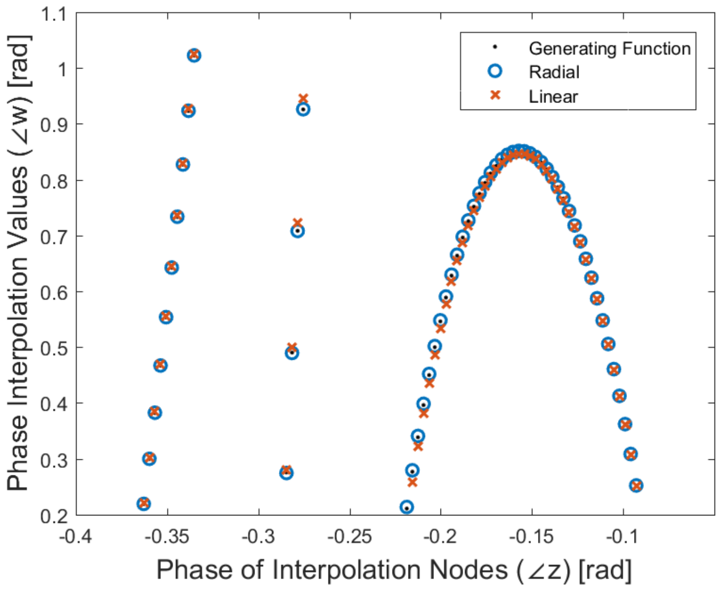


Figure 4-18: Comparison of the phases of the frequency response of the generating function and interpolants found by different continuation trajectories with similar interpolation error e_1. The radial path a solution that matches the phase of the generating function, while the linear path deviates at some nodes.

solution radially since then $\mathbf{w}(t)$ will remain the furthest from $|z| = 1$.

Finally, Figure 4-15 and Figure 4-16 show that the interpolation error e_1 of the radial path is about two orders of magnitude lower than that of the linear path — 5.73×10^{-4} and 4.12×10^{-2} for $\rho = 1.72 \times 10^{-12}$ — with only an order of magnitude increase in the magnitude error e_2.

4-3-3 Performance

This section analyzes the performance of the defocusing iterative solver (Algorithm 3 pg. 26) with a radial trajectory. The solver was evaluated for a hyperbolic secant generating function with $m = 4096$ interpolation points, and $\rho = 1.72 \times 10^{-12}$ maximum reflectivity.

The evolution of the interpolation error e_1 and magnitude error e_2 with respect to the iterations n for Algorithm 3 (pg. 26), illustrated by Figure 4-19 and Figure 4-20 respectively, approximates

$$10^{e_1} = r_1 2^n \quad \text{and} \quad 10^{e_2} = r_2 2^n \quad \text{for certain constants } r_1 \text{ and } r_2.$$

Interestingly, this suggest that it might possible to estimate the quality of the solution at an arbitrary iteration after a few iterations. However, the amount of iterations required to achieve a desired e_1 might not be feasible for real-time applications.

Moreover, Figure 4-21 shows that there a trade-off between e_1 and e_2, as increasing the number of iterations reduces e_1 but increases e_2. This trade-off is caused by finite numerical precision since, in theory, $e_2 = 0$. This means that Algorithm 3 (pg. 26) cannot reduce e_1 to

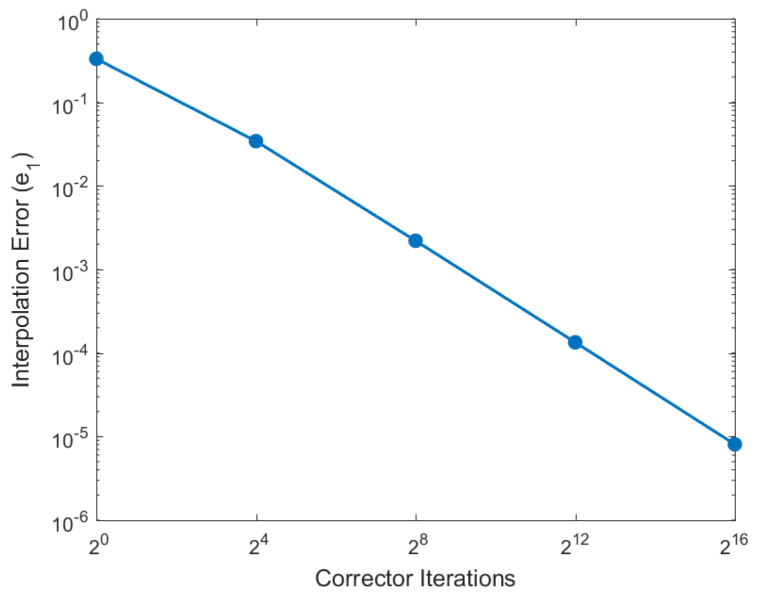


Figure 4-19: Interpolation error e_1 with respect to the number of corrector iterations of Algorithm 3 (pg. 26). There are diminishing returns on e_1 for increasing the number of iterations.

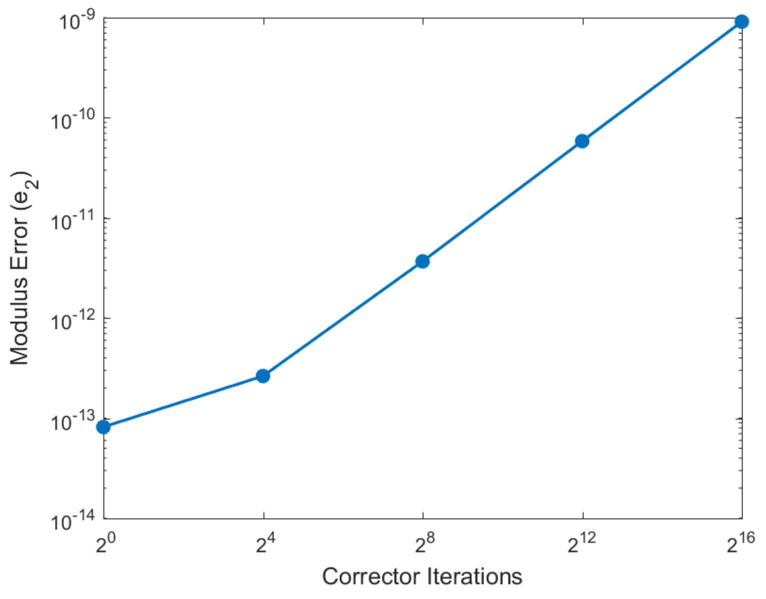


Figure 4-20: Magnitude error e_2 with respect to the number of corrector iterations of Algorithm 3 (pg. 26). The magnitude error e_2 increases with the number of iterations.

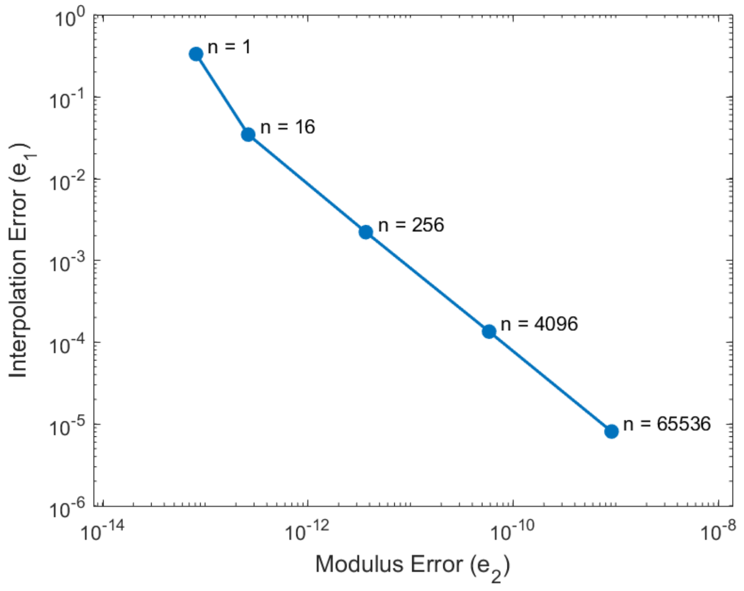


Figure 4-21: Magnitude error e_2 with respect to interpolation error e_1 after n iterations of Algorithm 3 (pg. 26). Reducing the interpolation error e_1 increases the magnitude error e_2, so there is a trade-off between the performance metrics.

an arbitrarily small value, since the maximum viable e_2 for layer peeling (largest e_2 for which layer peeling works as intended) place a bound on the achievable e_1.

4-3-4 Comparison to Conventional Solvers

This section presents comparisons between the defocusing iterative solver (Algorithm 3 pg. 26) and conventional techniques.

Comparing the solvers to conventional guaranteed techniques is not possible for interpolation data similar to a real application of non-linear Fourier transform, since the solvers available in the literature — a Nevanlinna-Pick interpolation with degree constraint by Nagamune and Blomqvist [12], and a boundary interpolation solver by Baratchart et al. [11] — assume the interpolation data is *self-conjugate* (symmetric with respect to the real line). Assuming self-conjugate data simplifies the problem significantly since the interpolant polynomials will have real coefficients [12][11], which is not guaranteed for synthesis. Furthermore, conventional solvers are designed for applications with few (< 100) interpolation points. Nevertheless, a comparison between Algorithm 3 (pg. 26) and Nagamune et al. solver is presented for interpolation data suitable for computation by both algorithms. The solvers where evaluated with respect to the number of interpolation points $m = 2^1, 2^2, \dots, 2^6$. After this, Nagamune's solver stopped converging consistently. The solvers were evaluated for 1000 transfer functions of McMillan degree $n = 1$ with random real coefficients for every m to get an adequate understanding of the performance of the algorithms in a broad set of problems, and to avoid data that benefits a solver over the other.

The interpolation error e_1 increases with the number of interpolation points for both algorithms, but Figure 4-22 shows that the rate of change of Nagamune's solver is exponential,

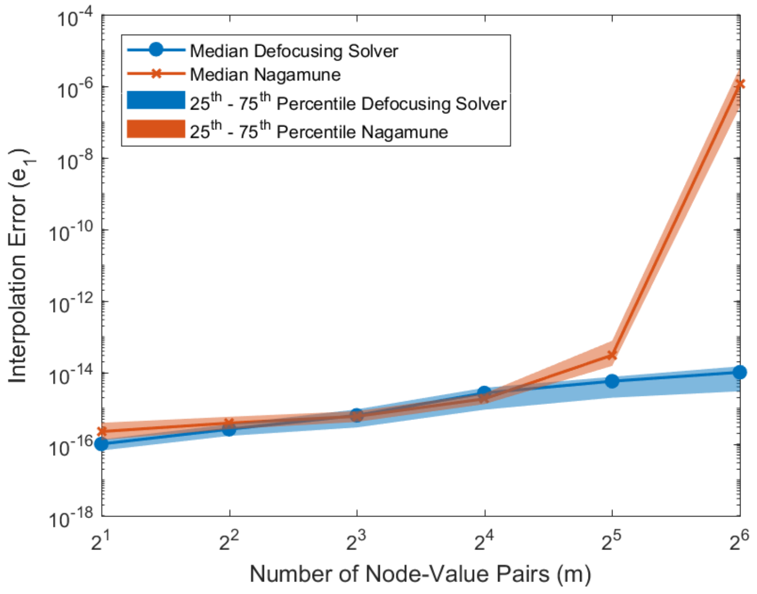


Figure 4-22: Statistical comparison of the interpolation error e_1 between a conventional Nevanlinna-Pick interpolation with degree constraint solver by Nagamune and Blomqvist, and the defocusing iterative solver (Algorithm 3 pg. 26). The test data was generated by sampling $m = 2^1, 2^2, \ldots, 2^6$ interpolation node-value pairs from 1000 transfer functions of McMillan degree $n = 1$ with random real coefficients. The interpolation error for Nevanlinna-Pick interpolation with degree constraint solver increases exponentially with respect to the quantity of interpolation point m, while increasing only linearly for Algorithm 3 (pg. 26).

while the defocusing iterative algorithm increases linearly. On the other hand, the modulus error e_2 increases very little for both solvers until 2^6 points, when Nagamunes's explodes. The defocusing iterative algorithm (Algorithm 3 pg. 26) finds solutions with comparable median e_2, but lower variance, as seen in Figure 4-23. Furthermore, the improvement in evaluation metrics comes with a significant reduction in complexity of the algorithm, illustrated by Figure 4-24, and the ability to handle a higher number of interpolation points.

The most interesting improvement is the reduction in interpolation error e_1, since Nagamune's solver is similar to the focusing iterative solver (Algorithm 5 pg. 28) and, in theory, $e_1 = 0$. This highlights the importance of reducing complexity in analytic interpolation since it also improves performance metrics by reducing the accumulation of errors from performing more operations with limited numerical precision.

A comparison between the defocusing iterative solver (Algorithm 3 pg. 26) and iterative synthesis (Algorithm 2 pg. 26) with respect to $\mathcal{Q} = 3, 3.5, \ldots, 7$ for a hyperbolic secant continuous spectrum with $m = 4096$ interpolation points and a stopping criteria of 41000 iterations, shows that Algorithm 3 (pg. 26) outperforms Algorithm 2 (pg. 26) for numerically difficult problems with $\rho < 10^{-9}$, but that for simple problems Algorithm 2 (pg. 26) achieves better e_1, illustrated by Figure 4-25. However Figure 4-25 and Figure 4-26 show that for comparable e_1 (in the range $10^{-11} < \rho < 10^{-9}$), Algorithm 3 (pg. 26) achieves lower e_2, and that Algorithm 2 (pg. 26) does not work for very difficult problems ($\rho < 10^{-11}$). This is due to the radial path approaching the solution from a numerically beneficial trajectory, which

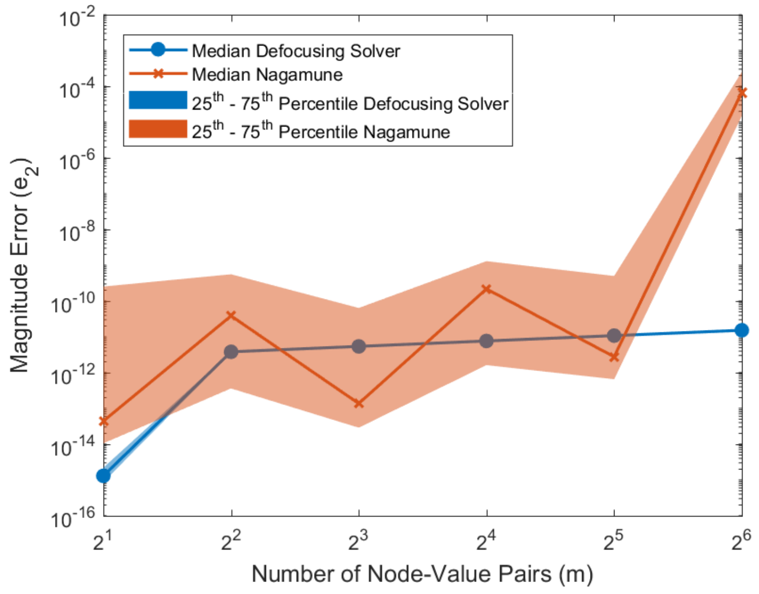


Figure 4-23: Statistical comparison of the magnitude error e_2 between a conventional Nevanlinna-Pick interpolation with degree constraint solver by Nagamune and Blomqvist, and the defocusing iterative solver (Algorithm 3 pg. 26). The test data was generated by sampling $m = 2^1, 2^2, \ldots, 2^6$ interpolation node-value pairs from 1000 transfer functions of McMillan degree $n = 1$ with random real coefficients. The median e_2 is comparable but the Nevanlinna-Pick interpolation with degree constraint solver has a higher variance than Algorithm 3 (pg. 26).

the errors in the spectral factorization step for problems with high ρ. This issue is further explored in Chapter A. A hybrid non-guaranteed technique that uses iterative synthesis as a corrector steps improves on the improvement speed of the solver with Newton root finding and has better performance for very difficult problems than iterative synthesis on its own, as seen in Figure 4-25 and Figure 4-26.

4-4 The Focusing Case

4-4-1 Comparison of Closed-Form Approximations

Just as for the defocusing case, four different heuristics for the phase of the point-value representation $\angle \mathbf{a}(\mathbf{z})$ were evaluated with respect to the number of interpolation node-value pairs m and inverse maximum amplitude of the continuous spectrum ρ. The test cases were:

- Matching the phase of the interpolation values: $\angle \mathbf{a}(\mathbf{z}) = \angle \mathbf{w}$.
- Constant shift in the phase of the interpolation values: $\angle \mathbf{a}(\mathbf{z}) = \angle \mathbf{w} + \pi/4$.
- Zero phase: $\angle \mathbf{a}(\mathbf{z}) = 0$.
- Random phase: $\angle \mathbf{a}(\mathbf{z}) =$ uniform random variable in $[0, 2\pi]$.

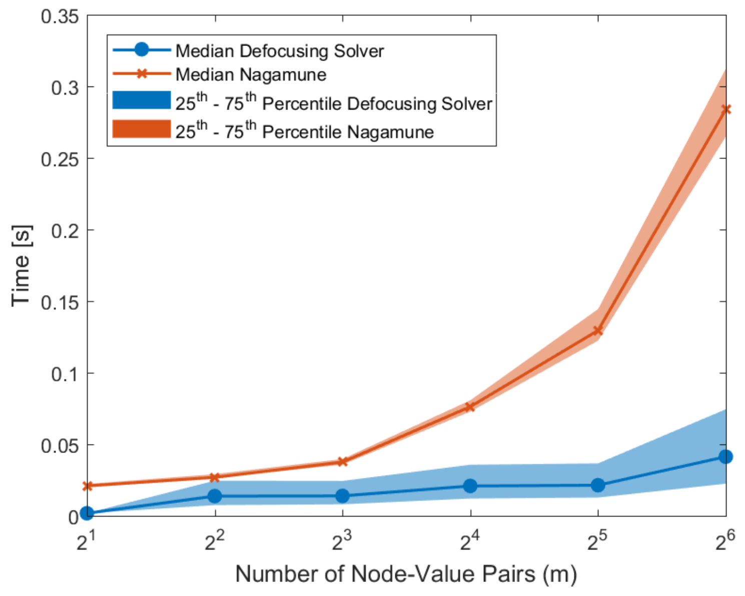


Figure 4-24: Statistical comparison of the convergence time between a conventional Nevanlinna-Pick interpolation with degree constraint solver by Nagamune and Blomqvist, and the defocusing iterative solver (Algorithm 3 pg. 26). The algorithm convergences when the interpolation error e_1 stops improving by less than 10^{-8}. The test data was generated by sampling $m = 2^1, 2^2, \ldots, 2^6$ interpolation node-value pairs from 1000 transfer functions of McMillan degree $n = 1$ with random real coefficients. The defocusing iterative has lower computational complexity.

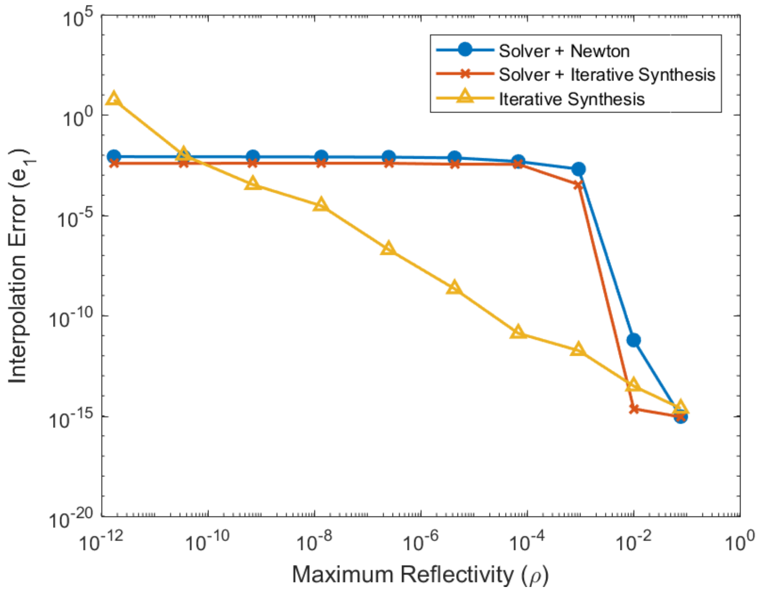


Figure 4-25: Comparison of the interpolation error e_1 between iterative synthesis (Algorithm 2 pg. 26) and the defocusing iterative solver (Algorithm 3 pg. 26). Algorithm 3 (pg. 26) achieves a lower e_1 for numerically difficult problems. The e_1 errors for the solver become constant because it reached the maximimum number of iterations allowed (41000)

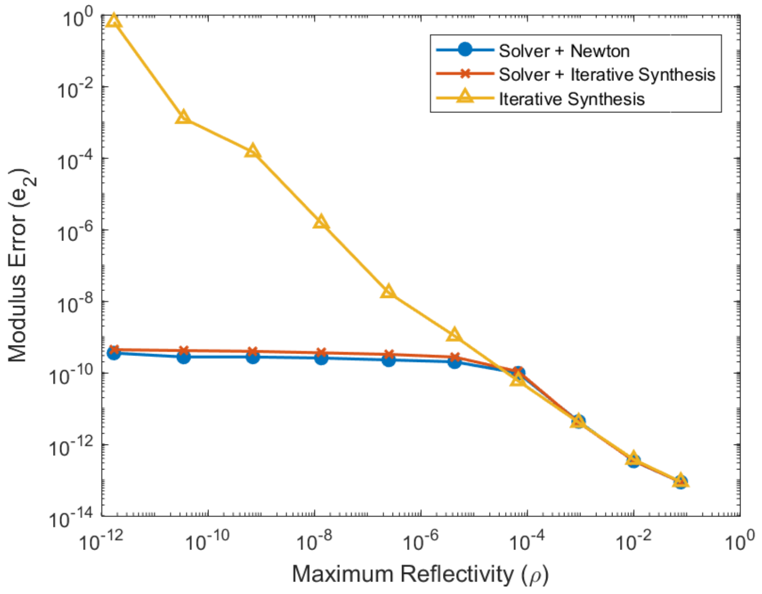


Figure 4-26: Comparison of the magnitude error e_2 between iterative synthesis (Algorithm 2 pg. 26) and the defocusing iterative solver (Algorithm 3 pg. 26). Algorithm 3 (pg. 26) achieves lower e_2 for all maximum reflectivity ρ. Furthermore, Algorithm 2 (pg. 26) is not viable for very difficult problems ($\rho < 10^{-11}$).

The approximations were compared for the focusing raised cosines, with $m = 2^8, 2^9, \ldots, 2^{20}$ interpolation node-value pairs and $\rho = 2^2, 2^1, \ldots, 2^{-8}$ maximum reflectivity.

The focusing explicit approximation (Algorithm 4 pg. 27) that better satisfies Eq. (2-11) is making $\angle a(z_k) = 0$, as it achieves a lower e_2 for any m for both simple and difficult problems, illustrated by Figure 4-31 and Figure 4-32 respectively. But Figure 4-33 and Figure 4-34 show that this advantage decreases as $\rho \to 0$. This matches the results of the defocusing experiments — where I found that the closest defocusing approximation is $\angle b(z_k) = \angle w_k$ — and implies that, at least for simple problems, $b(z)$ is responsible for most of the interpolant's phase.

There is little evidence of the numerical inaccuracies experienced by the defocusing explicit approximations (Algorithm 1 pg. 25) for increasing m, seen in Figure 4-27 and Figure 4-28, or for lowering ρ, seen in Figure 4-29 and Figure 4-30. Although e_1 changes with respect to m and ρ, it remains in the order of 10^{-16} — the same as Matlab's numerical precision.

Again, the maximum reflectivity ρ has a larger effect than the number of interpolation points m on the quality of the approximation. However the increase of e_2 for increasing m means that the iterative solver requires more iterations to reach the same e_2 for larger problems, assuming the rate of change is independent of m.

4-4-2 Comparison to Conventional Solvers

This section compares the focusing iterative solver (Algorithm 5 pg. 28) and *direct synthesis* [7] ($\angle \mathbf{a}(\mathbf{z}) = 0$ in Algorithm 4 pg. 27) with respect ρ on an eight raised cosines generating

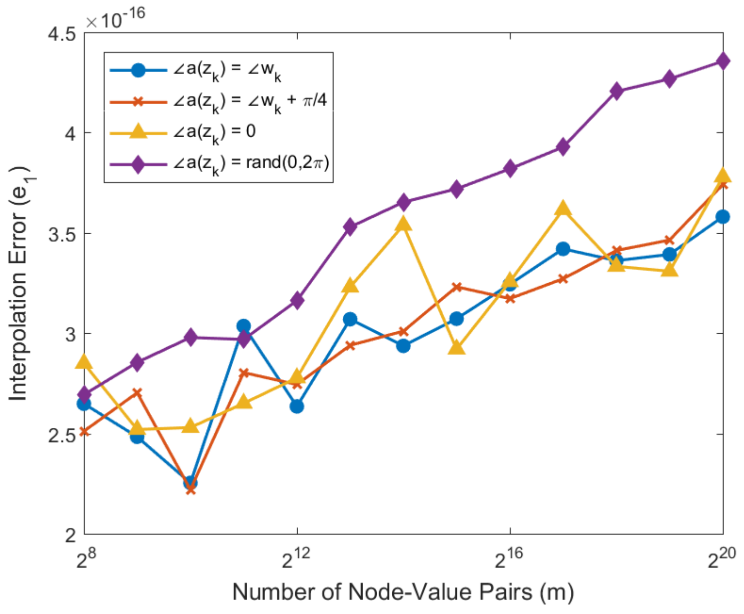


Figure 4-27: Comparison of the interpolation error e_1 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the number of interpolation points m using an eight raised cosines with inverted maximum amplitude $\rho = 2^2$ as the generating function. There is very little relationship between the parameterizing phase $\angle\mathbf{a}(\mathbf{z})$ and e_1. Furthermore, increasing m results in higher e_1, but the slope is small.

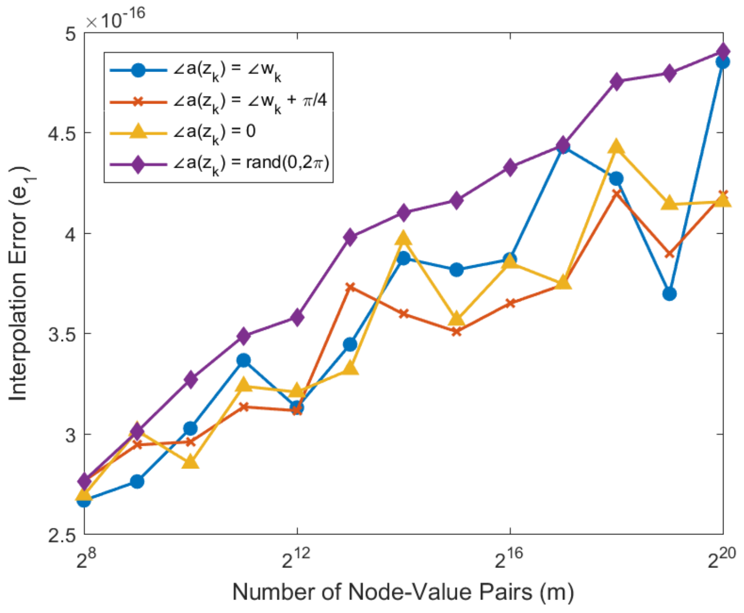


Figure 4-28: Comparison of the interpolation error e_1 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the number of interpolation points m using an eight raised cosines with inverted maximum amplitude $\rho = 2^{-8}$ as the generating function. There is very little relationship between the parameterizing phase $\angle\mathbf{a}(\mathbf{z})$ and e_1. Furthermore, increasing m results in higher e_1, but the slope is small.

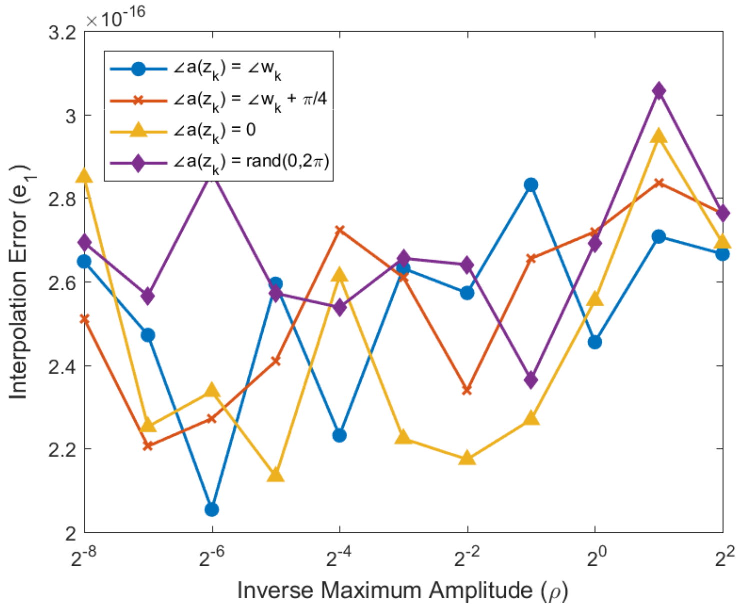


Figure 4-29: Comparison of the interpolation error e_1 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the inverse maximum amplitude ρ using an eight raised cosines with $m = 2^8$ interpolation points as the generating function. The parameterizing phase $\angle\mathbf{a}(\mathbf{z})$ or ρ have no noticeable effect on e_1.

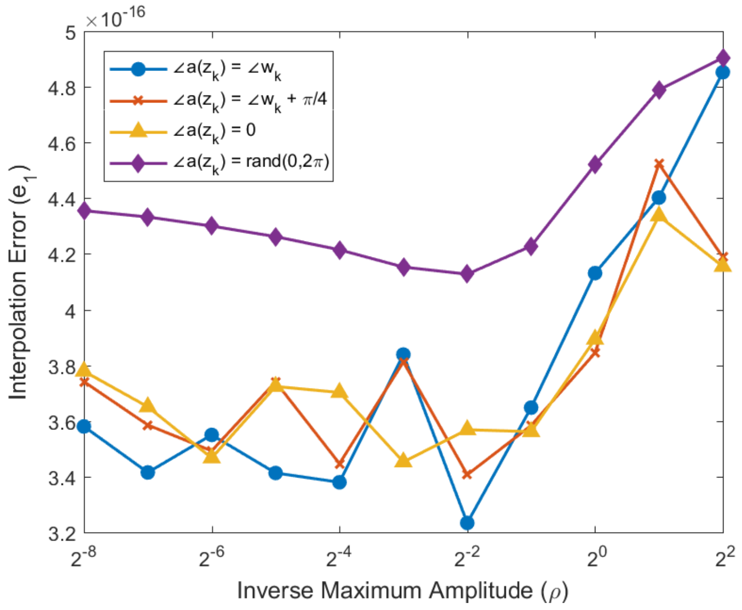


Figure 4-30: Comparison of the interpolation error e_1 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the inverse maximum amplitude ρ using an eight raised cosines with $m = 2^{20}$ interpolation points as the generating function. The parameterizing phase $\angle\mathbf{a}(\mathbf{z})$ or ρ have no noticeable effect on e_1.

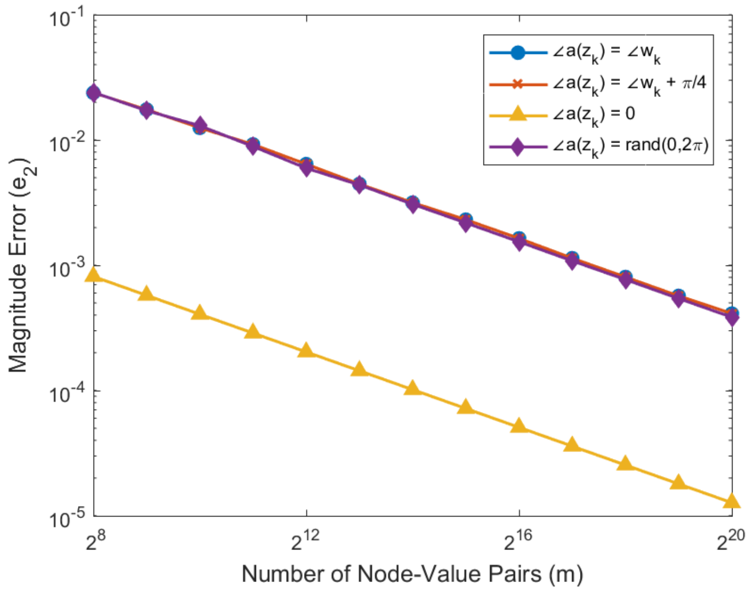


Figure 4-31: Comparison of the magnitude error e_2 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the number of interpolation points m using an eight raised cosines with inverted maximum amplitude $\rho = 2^2$ as the generating function. The magnitude error e_2 is inversely proportional to m. Furthermore, making the point-value representation real, i.e. $\angle \mathbf{a}(\mathbf{z}) = 0$, results in lower e_2 for any ρ.

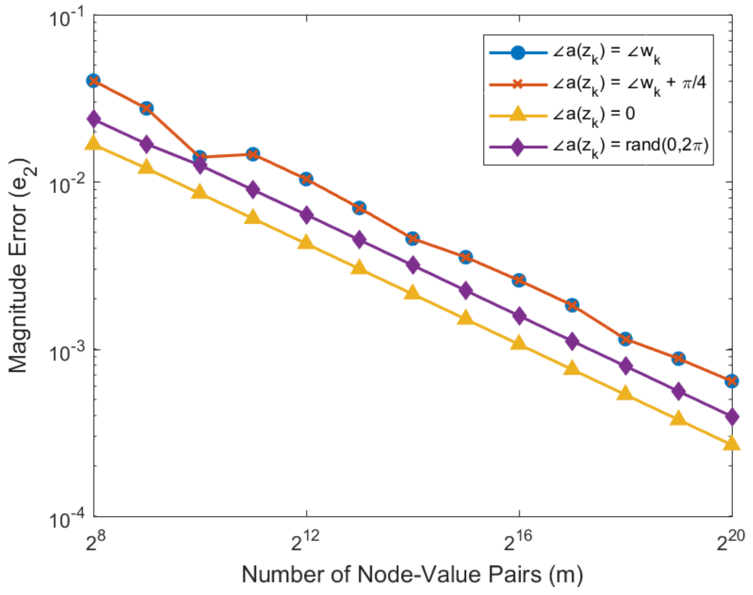


Figure 4-32: Comparison of the magnitude error e_2 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the number of interpolation points m using an eight raised cosines with inverted maximum amplitude $\rho = 2^{-8}$ as the generating function. The magnitude error e_2 is inversely proportional to m, and relatively high (> 1) for all the evaluated parameterizations. Moreover, making the point-value representation real, i.e. $\angle \mathbf{a}(\mathbf{z}) = 0$, results in slightly lower e_2 for any ρ.

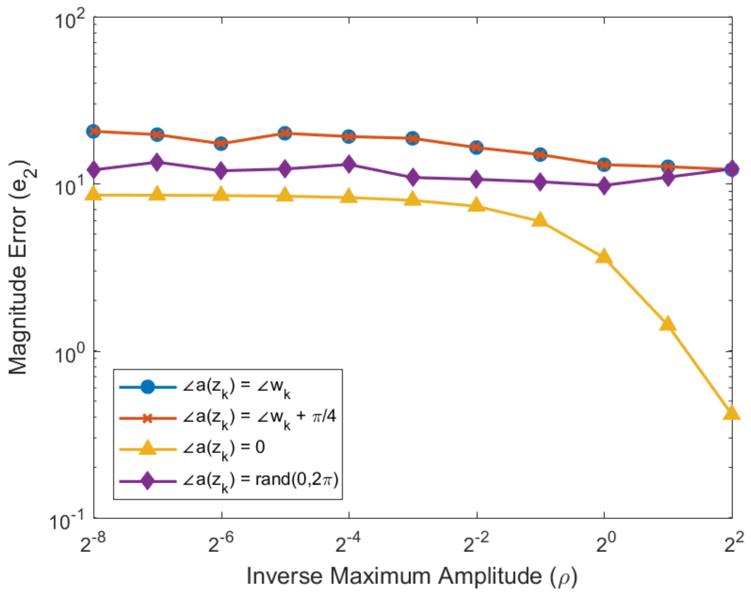


Figure 4-33: Comparison of the magnitude error e_2 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the inverse maximum amplitude ρ using an eight raised cosines with $m = 2^8$ interpolation points as the generating function. The parameterizing phase $\angle \mathbf{a}(\mathbf{z}) = 0$ results in a lower e_2 for all ρ, although the effect gets increasingly smaller until it plateaus at $\rho < 2^{-2}$.

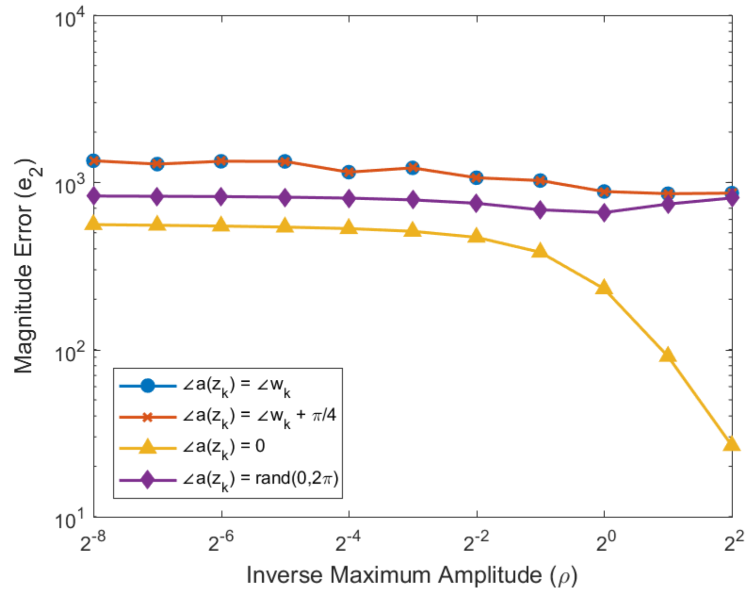


Figure 4-34: Comparison of the magnitude error e_2 of focusing explicit approximations (Algorithm 4 pg. 27) with respect to the inverse maximum amplitude ρ using an eight raised cosines with $m = 2^{20}$ interpolation points as the generating function. The parameterizing phase $\angle \mathbf{a}(\mathbf{z}) = 0$ results in a lower, but still relatively high, e_2 for all ρ, although the effect gets increasingly smaller until it plateaus at $\rho < 2^{-2}$.

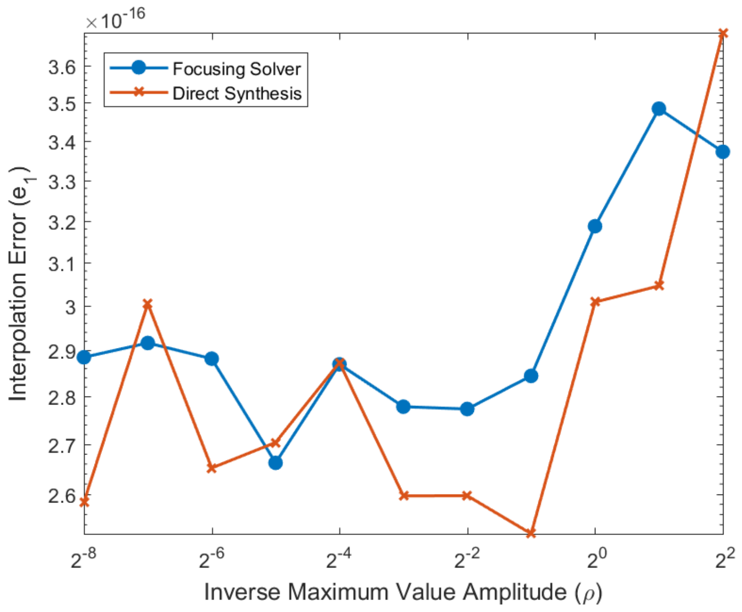


Figure 4-35: Comparison of the interpolation error e_1 between direct synthesis ($\angle \mathbf{a}(\mathbf{z}) = 0$ in Algorithm 4 pg. 27) and the focusing iterative solver (Algorithm 5 pg. 28). Both algorithms achieve similar e_1 — in the same order as Matlab's numerical precision.

function with $m = 2^{12}$ interpolation node-value pairs. The corrector step was set to stop once the rate of change in e_2 became less than 10^{-8}.

Figure 4-36 shows that the focusing iterative solver achieves (Algorithm 5 pg. 28) a significant reduction in magnitude error e_2 for simple problems (large ρ) but only improves more difficult problems by about an order of magnitude: from 1.66 to 1.07×10^{-5} for $\rho = 2^2$ and 34.93 to 3.07 for $\rho = 2^{-8}$. This means that it is not practical to obtain an arbitrary e_2 as the stopping criteria — the rate of change in e_2 is less than 10^{-8} — is reached relatively soon. However, Algorithm 5 (pg. 28) solver still represents an improvement over direct synthesis.

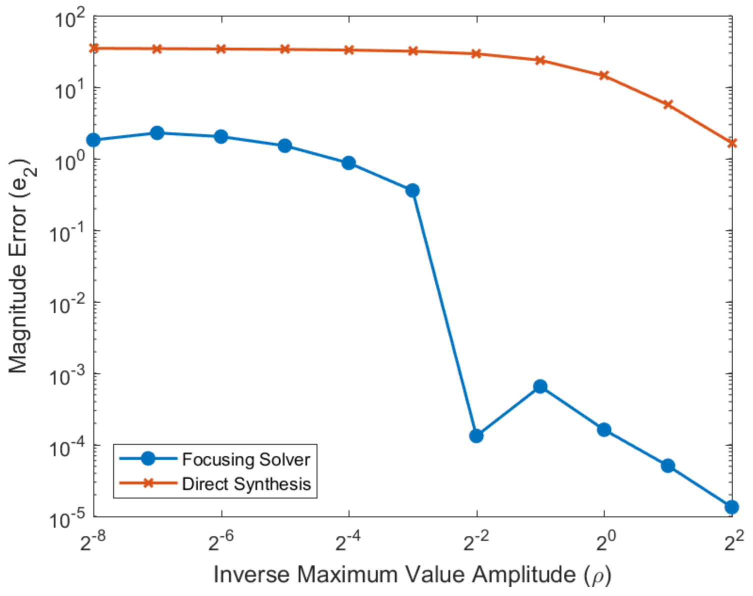


Figure 4-36: Comparison of the magnitude error e_2 between direct synthesis ($\angle \mathbf{a}(\mathbf{z}) = 0$ in Algorithm 4 pg. 27) and the focusing iterative solver (Algorithm 5 pg. 28).Algorithm 5 (pg. 28) achieves a lower e_2 for all values of ρ, but the difference gets smaller for more difficult problems.

Chapter 5

Conclusions and Future Work

5-1 Conclusions

Numerical continuation based on boundary Nevanlinna-Pick interpolation with degree constraint is a guaranteed solution to synthesis: it results in a solution with an arbitrary interpolation error e_1 in the defocusing case, or a modulus error e_2 that is better than current heuristic methods in the focusing case. The numerical guarantee is given by the fact that the corrector step is guaranteed to converge to a solution for a sufficiently small step-length in the predictor step, so the problem is convex over the continuation trajectory. However, this guarantee does not extend to finite-precision, as the numerical implementation has a trade-off between e_1 and e_2 that does not exist for infinite-precision operations. This means that it is not possible to achieve arbitrarily small e_1 and e_2 simultaneously.

Moreover, the solvers may not be effective for real-time applications, as the high-demands in error precision required for layer peeling means that the algorithms must perform many iterations, which translates into high computational cost, even if the cost of each iteration is low. This seems to be a limitation of the theory as the achievable errors are proportional to the number of iterations. Also, the solvers presented in this thesis have lower complexity than other guaranteed Nevanlinna-Pick interpolation with degree constraint solvers even when they result in solutions with lower e_1 and e_2.

The defocusing algorithms are generalizations of the boundary Nevanlinna-Pick interpolation with degree constraint solvers by Nagamune et al. and Baratchart et al. to interpolation data that is not necessarily self-conjugate. They are also less computationally complex and are able to handle more difficult problems and more interpolation points. The focusing algorithms are not generalization of conventional solvers because the interpolant is not bounded — a key condition of Nevanlinna-Pick interpolation — but they are an extension of Nevanlinna-Pick interpolation with degree constraint theory into a different type of analytic interpolation.

There are two key problems with the algorithms: first, when the local interpolation values $\mathbf{w}(t)$ are very close to the region of analyticity the predictor step may overshoot and push the polynomials outside this region which causes the algorithms to diverge; second, the corrector step stops improving significantly very quickly for complex problems.

In the defocusing case, the algorithm gets around these problems by approaching the solution in a radial path (from the origin of the complex plan). The radial path approaches the solution from the trajectory farthest away from the boundary minimizing the risk of overshooting the region of analyticity. Furthermore, the zero approximation (Eq. (3-1)), is the closest heuristic approximation for difficult problems.

There is no apparent solution to the lack of quick convergence in the focusing case. On the other hand, stopping the polynomials from moving away from the region of analyticity is done by reducing the predictor step-length when the point value representation of the polynomials $\mathbf{a}(\mathbf{z}), \mathbf{b}(\mathbf{z}) > 1$.

5-2 Future Work

5-2-1 Improvements to Explicit Approximations

The explicit approximations (Algorithm 1 and Algorithm 4) are perhaps the most interesting solutions for real-time applications since they require no iterations. Algorithm 4, in particular, may provide a way to compute an exact solution in closed-form. The main problem right now is that this approximation only satisfies Eq. (2-11) for $\mathbf{z} \in \mathbb{C}^m$. The reason is that Eq. (2-11) requires defining the polynomials over the over-sampled nodes $\mathbf{x} \in \mathbb{C}^{2m-1}$ and we only know the value of the interpolation values at $\mathbf{z}$, so the full over-sampled representation is undefined. However, assuming the interpolation values are given by a function $\mathbf{f}(\mathbf{z}) := \mathbf{w}$, then the over-sampled values are simply $\mathbf{f}(\mathbf{x})$. Now the problem is finding a function $f(z)$ such that computing $a(z)$ from the over-sampled point-value representation $\mathbf{a}(\mathbf{x}) \in \mathbb{C}^{2m-1}$ results in polynomials of degree m.

A similar exact solution may exist for Algorithm 1. The explicit approximations are parameterized by the angles of the point-value representation $\angle\mathbf{b}(\mathbf{z})$, and one of these parameterizations leads to the exact solution. Since Eq. (3-6) also has a closed-form solution [33] there may be a explicit inversion. A first step would be finding out how spectral factorization maps $\angle\mathbf{b}(\mathbf{z}) \rightarrow \angle\mathbf{a}(\mathbf{z})$.

5-2-2 Improvements to the Iterative Solvers

Choosing the continuation trajectory wisely in the defocusing solver leads to considerable improvement. It may be that there is a trajectory other than a radial path for the focusing algorithm which yields better results. Furthermore, there might be a way to combine early solutions of the defocusing iterative solver (Algorithm 3) using a radial path — which interpolates the phases very precisely — with the explicit algorithm to quickly find the solution.

5-2-3 Extending the Solvers to Other Applications

An interesting research direction would be to extend the algorithm to other applications. The defocusing iterative solver in particular is intuitively simpler than the Nevanlinna-Pick interpolation with degree constraint algorithms by Byrnes et al. [10] and offers the same guarantees when the interpolation nodes are at the unit circle $|z| = 1$. It would be interesting to extend the solvers to include interpolation nodes in the unit disk $|z| \leq 1$, a requirement for internal stability in control [21]. At the moment, the solver can be used in the impedance matching problem by Baratchart et al. [11] by implementing an additional conformal map [14] between the interpolation node-value pairs.

Furthermore, the algorithm compares very favorably with Nagamune and Blomqvist's solver [12] — the only freely available solver for boundary Nevanlinna-Pick interpolation with degree constraint — despite being designed for a more general type of problems. It would be worth it to extend some of the Nevanlinna-Pick interpolation with degree constraint features, like derivative constraints for the interpolation values and interpolation of matrix polynomials.

Exploring minimum degree interpolants is outside the scope of this thesis — layer peeling requires the degree to match the interpolation points — but it is useful for other applications. The relationship to identification might provide a way forward, since exploring the accuracy of the interpolant (model) with respect to its order is a common task [34]. Another idea is to move away from exact to optimal interpolation, where the interpolant minimizes a cost metric. This would mean that Eq. (2-10) does not define a linear map between polynomials of degree $m-1$, but a *linear least-squares* problem [34] between polynomials of arbitrary degree.

5-2-4 Interaction with the non-linear Fourier transform

There might be a faster solution to synthesis by moving away from Problem 1, but keeping the core idea intact. Instead of trying to interpolate the points perfectly the problem could be to optimally match the signal. The idea here is to follow the procedure of the focusing explicit algorithm (Algorithm 5, pg. 28) and compute the polynomials based on the over-sampled point-value representations. The difficulty would be finding interpolation values that result in polynomials of degree $m-1$ and not $2m-1$, as would normally be the case for the over-sampled representation. An optimal approach would allow the leeway to satisfy this condition and still match the frequency response of the spectrum.

Appendix A

Comparison of spectral factorization algorithms

In this appendix I compare four spectral factorization algorithms. This is the largest source of numerical error in the solver algorithms. The solver uses Kolmogorov algorithm for spectral factorization [29] as it is the fastest algorithm for large problems [35]. The other evaluated algorithms are Bauer [29], Weiner-Hopf [36], and Wilson-Burg [35].

A-1 Spectral factorization algorithms

A-1-1 Kolmogorov

Kolmogorov spectral factorization, sometimes known as Hilbert transform method [29], is based around the idea that the magnitude of the frequency response determines the response for a minimum phase system. The phase is

$$\angle a(z) = -\mathcal{H}\Big(\log|a(z)|\Big),$$

where $\mathcal{H}$ is the Hilbert transform [29]. The technique can be implemented using fast Fourier transforms and requires no iterations, which makes it the fastest spectral factorization algorithm.

A-1-2 Bauer

The idea behind Bauer's method [37] is to approximate the coefficients of $a(z)$ by computing the Cholesky decomposition of a finite Toeplitz matrix of size $k \times k$

$$\mathbf{T}_k = \begin{bmatrix} S_0 & S_{-1} & S_{-2} & \cdots \\ S_1 & S_0 & S_{-1} & \cdots \\ S_2 & S_1 & S_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{A-1}$$

First, compute the triangular decomposition $\mathbf{T}_k = \mathbf{L}_k \mathbf{D}_k \overline{\mathbf{L}_k}^T$ with $\mathbf{L}_k$ a lower triangular matrix with unit diagonal entries, and $\mathbf{D}_k$ a diagonal matrix. Then

$$d_k \mathbf{l}_k \to \mathbf{a} \quad \text{as } k \to \infty,$$

where d_k is the last value in $\mathbf{D}_k$ and $\mathbf{l}_k$ is the last row of $\mathbf{L}_k$.

For the numerical comparison, the matrix of Eq. (A-1) was of size $32 * m \times 32 * m$ — with m the number of coefficients of $a(z)$ — up to a maximum size of $2^14 \times 2^14$, after which the matrix became too large to store in RAM.

A-1-3 Wilson-Burg

Wilson-Burg algorithm solves the spectral factorization problem with Newton's iterative method

$$S(z) = a_t(z)\overline{a_{t+1}(z^{-1})} + \overline{a_t(z^{-1})}a_{t+1}(z) - a_t(z)\overline{a_t(z^{-1})}.$$

Diving by $a_t(z)\overline{a_t(z)}$ results in a convenient form where the terms can be separated into casual and anti-casual

$$1 + \frac{S(z)}{a_t(z)\overline{a_t(z^{-1})}} = \frac{a_{t+1}(z)}{a_t(z)} + \frac{\overline{a_{t+1}(z^{-1})}}{\overline{a_t(z^{-1})}}. \tag{A-2}$$

Two initial guesses were used: $a(z) = z^{m-1}$ as proposed by [35], and the result from Kolmogorov algorithm. The algorithm was limited to 100 iterations.

A-1-4 Wiener-Hopf

Wiener-Hopf algorithm [36] is based on solving a non-linear system arising from rewriting the spectral factorization problem in terms of the polynomial coefficients,

$$\mathbf{S} = \mathbf{T}\hat{\mathbf{a}}, \tag{A-3}$$

where

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_0 \\ \mathbf{S}_1 \end{bmatrix} = \left[\begin{array}{c} S_{-n} \\ S_{-n+1} \\ \vdots \\ S_1 \\ \hline S_0 \\ \vdots \\ S_n \end{array}\right], \quad \mathbf{T} = \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} = \left[\begin{array}{cccc} a_0 & & & \\ a_1 & a_0 & & \\ \vdots & \ddots & \ddots & \\ 1 & a_{n-1} & \cdots & a_0 \\ \hline & 1 & \ddots & a_1 \\ & & \ddots & \vdots \\ & & & 1 \end{array}\right], \quad \hat{\mathbf{a}} = \begin{bmatrix} a_n\overline{a_n} \\ a_n\overline{a_{n-1}} \\ \vdots \\ a_n\overline{a_0} \end{bmatrix}$$

into non-linear equation

$$\mathbf{S}_0 - \mathbf{T}_0\mathbf{T}_1^{-1}\mathbf{S}_1 = 0, \tag{A-4}$$

and using Newton root finding to solve Eq. (A-4).

Just as with Wilson-Burg, two initial guesses were used: an approximated solution to Eq. (A-4) proposed by [36], and the result from Kolmogorov algorithm. The algorithm was limited to 100 iterations.

A-2 Numerical Comparison

A-2-1 Two Bad Polynomials

This test was proposed by [36] as an example of a difficult spectral factorization problem. The numerical difficulty increases as the problem grows larger. The Laurent polynomial of order n is given by

$$S(z) = \Big(\frac{1}{n}(1 + z^{-1} + \cdots + z^{-n+1}) + z^{-n+1}\Big)\Big(n + z + z^2 + \cdots + z^n\Big).$$

The error was measured as $\|\hat{a}(z) - a(z)\|$, where $\hat{a}(z)$ is the result of the algorithms and $a(z)$ is the known solution.

Wiener-Hopf algorithm with the initial guess proposed by [36] delivers the best results, specially for larger and more difficult problems, as seen in Figure A-1. However, it is considerably more expensive than Kolmogorov's method, as seen in Figure A-2, although any comparison on timing should be taken with a grain of salt, as they may reflect differences in implementation, and circumstances, just as much as differences in the complexity of the underlying algorithms.

A-2-2 Spectral Factorization for Synthesis

This test measures the suitability of each algorithm for synthesis. The problem is to find a polynomial $a(z)$ that satisfies Eq. (2-11) for a given $b(z)$ and interpolation points from a

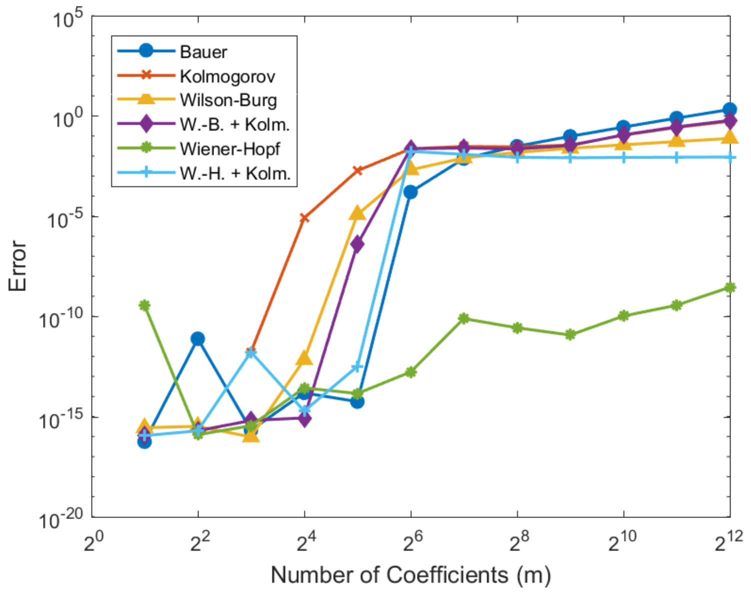


Figure A-1: Error of spectral factorization with respect to the number of coefficients of the minimum phase polynomial for several algorithms

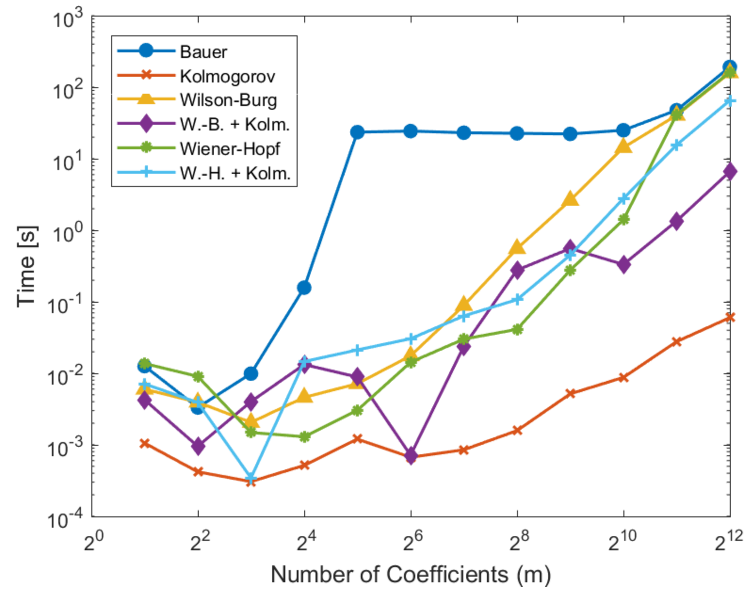


Figure A-2: Execution time, in seconds, for spectral factorization with respect to the number of coefficients of the minimum phase polynomial for several algorithms

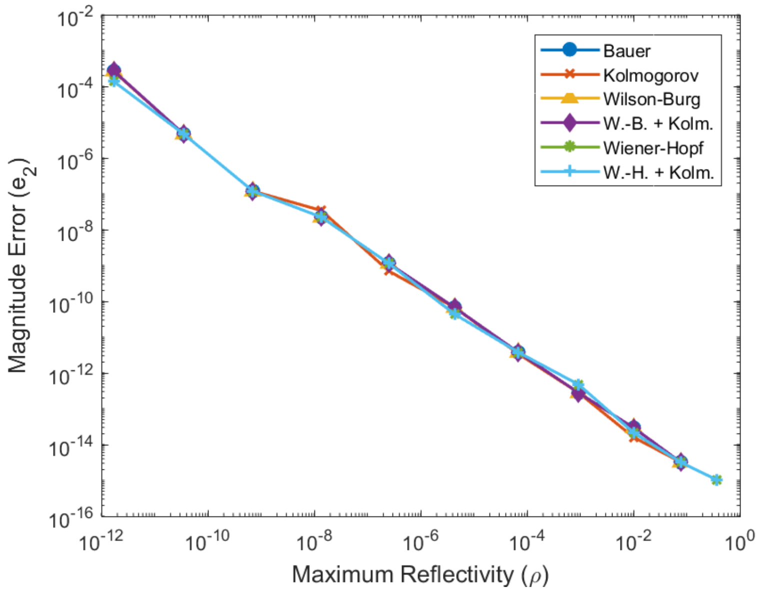


Figure A-3: Magnitude error e_2 with respect to maximum reflectivity ρ for a problem with $m = 2$ iteration points of several spectral factorization algorithms

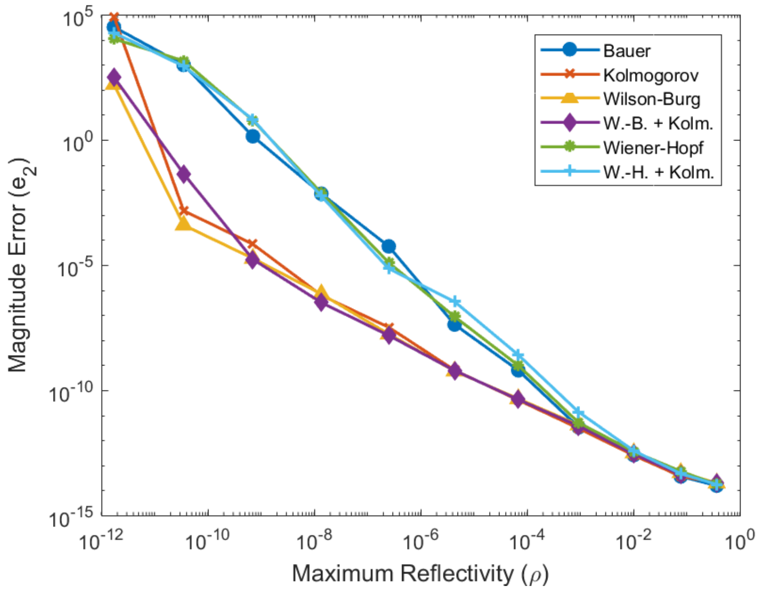


Figure A-4: Magnitude error e_2 with respect to maximum reflectivity ρ for a problem with $m = 256$ iteration points of several spectral factorization algorithms

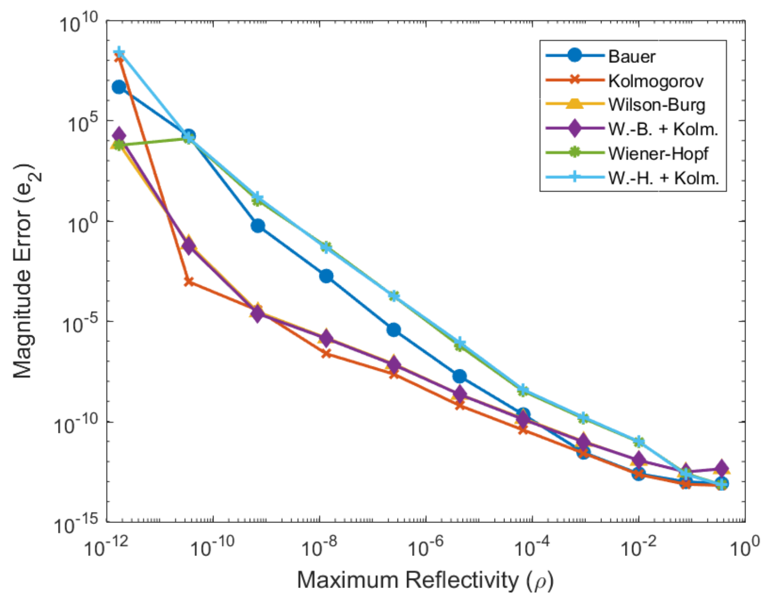


Figure A-5: Magnitude error e_2 with respect to maximum reflectivity ρ for a problem with $m = 4096$ iteration points of several spectral factorization algorithms

hyperbolic secant. The iterative algorithm, Wilson-Burg and Wiener-Hopf, are forced to take at least one iteration from its initial guess.

Wilson-Burg spectral factorization [35] is well suited for very ill-conditioned problems — problems with $\rho \leq 10^{-12}$ — and reaches an equal or better solution in this case, as seen in Figure A-3, Figure A-4, and Figure A-5. In any other case, it is preferable to use Kolmogorov since it has similar errors but much lower execution time. Starting the iterative algorithms with the result from Kolmogorov's offers no advantage, and even makes the result worse in some cases. This is caused by numerical innacuracies: Wiener-Hopf or Wilson-Burg iterate on monic representations (coefficient of largest degree is normalized to one), and even if they do not change the solution they perform operations on the polynomial, which generates numerical errors and makes the solution worse.

Appendix B

Mathematical Proofs

B-1 Proof that the Magnitude Condition is Finite-Dimensional

Define the Laurent polynomials

$$A(z) := |a(z)|^2 \quad \text{and} \quad B(z) := |b(z)|^2,$$

then for $a(z)$ with fixed degree m, Eq. (2-11) is

$$A(z) + \gamma B(z) = 1 \quad \text{for } |z| = 1.$$

By comparing the coefficients with the same powers we get

$$\begin{aligned}
A_{-m+1} + \gamma B_{-m+1} &= 0, \\
&\vdots \\
A_{-2} + \gamma B_{-2} &= 0, \\
A_{-1} + \gamma B_{-1} &= 0, \\
A_0 + \gamma B_0 &= 1, \\
A_1 + \gamma B_1 &= 0, \\
A_2 + \gamma B_2 &= 0, \\
&\vdots \\
A_{m-1} + \gamma B_{m-1} &= 0
\end{aligned}$$

which is a system of $2m - 1$ equations.

Appendix C

Matlab code

C-1 Defocusing Solver

```matlab
%% Load test data
fn = 'defocusing';
load(fn);

%% Set hyper-parameters
dt0 = 0.2;
dt1 = 0.01;

nM = size(Z,2);
for iM = 1:nM
    z = Z(:,iM);
    w = W(:,iM);

    [b, a, h] = solver(w, dt0, dt1, 'radial', fn);
    [interpError(iM), magError(iM)] = errors(b, a, w, fn);
end
```

```matlab
function [b, a, historicalData] = solver(w, dt0, dt1, path, caseK)
% SOLVER Finds the solution of a boundary analytic interpolation problem.
%
% Synopsis
%   [b, a, historicalData] = solver(w, dt0, dt1, path, caseK)
%
% Description
%   solver computes the denominator a and the numerator b of a rational
%   polynomial function that satisfies the modulus condition
%   |a(x)|^2+k|b(x)|^2=1, for all |x|=1, and the interpolation condition
%   b(z)/a(z) = w, where z are distinct equidistant nodes in the unit
%   circle.
%
% Inputs ([] are optional)
%   (vector) w: [m 1] vector of interpolation values the function should
%                 take at m distinct equidistant nodes on the unit circle.
%   (scalar) dt0: initial steplength.
%   (scalar) dt1: final steplength.
%   (string) path: string indicating the trajectory from initial
%                    approximation w' to the interpolation values w.
%                    Can be:
%            - 'Linear': default path. The values follow a linear path,
%              w(t) = w'+t(w-w').
%            - 'Circular': the magnitudes do not change and the phases
%              follow a linear path,
%              \angle w(t) = \angle w'+t(\angle w - \angle w'),
%              w(t) = |w|e^(i \angle w(t)).
%            - 'Linear': the values follow a linear path from zero,
%              w(t) = t(w).
%   (string) [caseK]: string indicating the case. Can be:
%            - 'Defocusing': default case. k = -1.
%            - 'Focusing': k = 1.
%
% Outputs
%   (vector) b: [m 1] vector representing the coefficients of the numerator
%               of a rational polynomial function of order n = m-1,
%               b = [b0, ..., bn]^T -> b0 + b1*z^(-1) + ... + bn*z^(n).
%   (vector) a: [m 1] vector representing the coefficients of the
%               denominator of a rational function of order n = m-1,
%               a = [a0, ..., an]^T -> a0 + a1*z^(-1) + ... an*z^(-n).
%   (struct) historicalData: data along the path. Contains:
%            - (scalar) .iter: number of predictor-corrector iterations.
%            - (vector) .interpolationError: [1 iter] vector of normalized
%                                            L2 interpolation errors at
%                                            each iteration.
%            - (vector) .magnitudeError: [1 iter] vector of L2 magnitude
%                                        condition errors at each iteration
%            - (vector) .b: [m iter] vector of b at each iteration.
%            - (vector) .a: [m iter] vector of a at each iteration.
%
% Examples
%   [b, a, h] = solver([0.5 0.3], 0.1, 0.01, 'linear', 'defocusing')
%
% Author
%   Julian Uribe Jaramillo

%% Organize inputs and set parameters
if (nargin < 5)
    caseK = 'defocusing'; % Default case
end

w = w(:);
path = lower(path);
```

```
caseK = lower(caseK);

m = length(w); % Number of interpolation points
corIter = 0; % Number of corrector iterations

maxErrorPath = 1e-5; % Maximum interpolation error along the path
maxErrorFinal = 1e-5; % Maximum interpolation error at the end point

%% Compute number of iterations
i = 0;
t = 0;
while (t < 1)
    i = i + 1;

    %% Update arclength (t)
    dt = dt0+t*(dt1-dt0); % Steplength (\alpha)

    if (t + dt > 1)
        t = 1; % Arclength (t)
    else
        t = t+dt;
    end
end
iter = i;

%% Initialization
[b, a, w0] = initialize(w, path, caseK); % Initial approximation

% Polar representation of the interpolation values (w)
magW = abs(w); % Magnitudes of interpolation values (|w|)
phaseW = mod(angle(w), 2*pi); % Phases of final w in [0, 2*pi] (\angle w)
phaseW0 = mod(angle(w0), 2*pi); % Phases of initial w (\angle w')

% Set size of historical data
interpolationError = zeros(1, iter); % Interpolation error (e1(t))
modulusError = zeros(1, iter); % Modulus error (e2(t))
bH = zeros(m, iter); % Historical numerator coefficients (b(z,t))
aH = zeros(m, iter); % Historical denominator coefficients (a(z,t))

%% Compute data for initial approximation
[interpolationError(1), modulusError(1)] = errors(b, a, w, caseK);
bH(:, 1) = b;
aH(:, 1) = a;

%% Iterate until end of arclength (t=1)
i = 0;
t = 0;
while (t < 1)
    %% Update the trajectory and find the local interpolation values
    [t, dt] = updateArclength(t, dt0, dt1);
    [wt, dw] = updatePath(w, w0, t, path);

    %% Predictor-corrector iteration
    [b, a] = predictor(b, a, wt, dw, dt, caseK);
    [b, a] = corrector(b, a, wt, maxErrorPath, caseK);

    %% Save historical data
    i = i+1;
    [interpolationError(i), modulusError(i)] = errors(b, a, w);
    bH(:, i) = b;
    aH(:, i) = a;
end

[b, a] = corrector(b, a, w, maxErrorFinal, caseK);

%% Organize outputs
b = b(:);
a = a(:);
historicalData.iter = iter;
historicalData.corIter = corIter;
historicalData.interpolationError = interpolationError;
historicalData.modulusError = modulusError;
historicalData.b = bH;
historicalData.a = aH;

%% Subfuctions
    function [b, a, w0] = initialize(w, path, caseK)
        % INITIALIZE Finds an approximated solution to a boundary
           analytic
        % interpolation problem.

        switch (caseK)
            case 'focusing'
                k = 1;
            case 'defocusing'
                k = -1;
        end

        % Compute the initial polynomials
        if (strcmp(path, 'linear') || strcmp(path, 'circular'))
            B = w./(sqrt(1+k*abs(w).^2)); % Point-value rep. of b(z)
            b = ifft(B); % Coefficient representation of b(z)

            a = b2a(b, caseK); % Coefficient representation of a(z)
            A = fft(a); % Point-value representation of a(z)

            w0 = B./A; % Initial interpolation values

        elseif strcmp(path, 'radial')
            b = zeros(m, 1);
            a = b2a(b, caseK);

            w0 = zeros(m, 1);

        else
            error('Path should be "Linear", "Circular" or "Radial"')
        end
    end

    function [t, dt] = updateArclength(t, dt0, dt1)
        % UPDATEARCLENGTH Updates the arclength t

        %% Update the steplength
        dt = dt0+t*(dt1-dt0); % Steplength (\alpha)

        %% Update the arclength and keep it in range [0, 1]
        if (t + dt > 1)
            t = 1; % Arclength (t)
        else
            t = t+dt;
        end
    end

    function [wt, dw] = updatePath(w, w0, t, path)
        % UPDATEPATH Computes the local interpolation values in the path

        %% Update path
        switch (path)
            case 'linear'
                wt = w0+t*(w-w0); % Local interpolation values (w(t))
                dw = w-w0; % Derivatives of wt wrt t (dw(t)/dt)

            case 'circular'
                phaseWt = phaseW0+t*(phaseW-phaseW0);
                phaseWt = mod(phaseWt, 2*pi);
                wt = magW.*exp(1i*phaseWt);
                dw = 1i*wt.*(phaseW-phaseW0);

            case 'radial'
                wt = t*w;
                dw = w;

            otherwise
                error('Path should be "Linear", "Circular" or "Radial"')
        end
    end

    function [b, a] = predictor(b, a, wt, dw, dt, caseK)
        % PREDICTOR Approximates the next solution

        switch (caseK)
            case 'focusing'
                k = 1;
            case 'defocusing'
                k = -1;
        end

        %% Compute point-value representation
        B = fft(b);
        A = fft(a);

        %% Update polynomials
        dB = (A.*dw)./(1+k*wt.*conj(B./A)); % Derivative wrt to t (db(z,t
            )/dt)
        B = B + dt*dB; % Point-value representation

        %% Compute the coefficient representations
        b = ifft(B);
        a = b2a(b, caseK);
    end

    function [b, a] = corrector(b, a, wt, maxError, caseK) % Newton
        % CORRECTOR Finds the local solution closest to the current
        % approximation

        B = fft(b);
        A = fft(a);
        H = B - wt.*A;
        error = norm(H);

        while (error > maxError)
            % Exit algorithm if over 20000 iterations
            if (corIter > 20000)
                break;
            end

            corIter = corIter + 1;
            switch (caseK)
                case 'focusing'
                    k = 1;
                case 'defocusing'
                    k = -1;
            end

            %% Compute point-value representation
            B = fft(b);
            A = fft(a);

            %% Update polynomials
            H = B - wt.*A;
            dH = 1 - k*wt.*conj(B./A); % Derivative wrt to t (db(z,t)/dt)

            oldB = B;
            B = B - H./dH; % Point-value representation

            % Exit corrector if change is less than machine precision
            if (norm(oldB - B) < eps)
                break;
            end

            %% Compute the coefficient representations
            b = ifft(B);
            a = b2a(b, caseK);

            error = norm(H);
        end
    end
end
```

```
function a = b2a(b, method, w)
% B2A Computes the denominator of a rational polynomial function.
%
% Synopsis
%   a = b2a(b, method, w)
%
% Description
%   b2a computes the denominator a from the numerator b of a rational
%   polynomial function. The polynomials satisfy either the modulus
%   condition |a(z)|^2-|b(z)|^2=1 for all |z|=1; or the interpolation
%   condition b(z_k)/a(z_k) = w_k.
%
% Inputs ([] are optional)
%   (vector) b: [m 1] vector representing the coefficients of the
    numerator
%               of a rational polynomial function of order n = m-1,
%               b = [b0, ..., bn]^T -> b0 + b1*z^(-1) + ... + bn*z^(n).
%   (string) [method]: string indicating the condition the function
%                      satisfies, can be:
%             - 'Modulus': default method. The polynomials satisfy
%               |a(x)|^2-|b(x)|^2=1 for all |x|=1.
%             - 'Interpolation': the polynomials satisfy b(z)/a(z) = w,
%               where z are m distinct equidistant nodes in the unit
    circle
%               and w are given interpolation values.
%   (vector) [w]: [m 1] vector of the values the function should take at
    m
%                   distinct equidistant nodes on the unit circle.
%
% Outputs
%   (vector) a: [m 1] vector representing the coefficients of the
%               denominator of a rational function of order n = m-1,
%               a = [a0, ..., an]^T -> a0 + a1*z^(-1) + ... an*z^(-n).
%
% Examples
%   a = b2a([1 0.3].', 'Interpolation', [0.4 0.3].')
%
% Author
%   Julian Uribe Jaramillo

%% Organize inputs and set parameters
if (nargin < 2)
    method = 'defocusing'; % Default
end

b = b(:);
```

```matlab
method = lower(method);
m = length(b); % Number of interpolation points

switch (method)
    case 'defocusing'
        bb = xcorr(b);

        aa = bb;
        aa(m) = aa(m) + 1;

        a = spec(aa);
        a = a(:);

    case 'focusing'
        bb = xcorr(b);

        aa = -bb;
        aa(m) = 1 - aa(m);

        a = spec(aa);
        a = a(:);

    case 'interpolation'
        warning('Might result in an unstable interpolant')
        if (nargin < 3)
            error('Interpolation values missing')
        else
            B = fft(b);
            A = B./w;
            a = ifft(A);
        end

    otherwise
        error('Method should be "Defocusing", "Focusing" or "
           Interpolation"')
end
end
```

```matlab
function s = spec(pp)
% Polynomial spectral factorization

pp = pp(:).';
length_p = 0.5*(length(pp)+1);

m = 32*length_p;
pp = pp(length_p:end);
R = fft([pp zeros(1, 2*m-2*length_p+1) conj(pp(length_p:-1:2))]);
R = real(R);

rho = min(R);

scl = sqrt(eps)*max(R);

```

```
if (rho <= -scl)
    warning('pp(z)=%g (<=0) on the unit circle', min(R));
elseif (rho <= 0)
    idx = (R <= 0);
    R(idx) = R(idx) + scl;
end

x = 0.5*log(R);

X = fft(x);
X(1) = 0;
X(m) = 0;
X(2:m-1) = -1i*X(2:m-1);
X(m+1:end) = 1i*X(m+1:end);

y = ifft(X);

H = exp(x-1i*y);

h = ifft(H);

s = conj(h(length_p:-1:1));
s = s(:);

end
```

```
function [interpolationError, modulusError] = errors(b, a, w, caseK)
% ERRORS Compute the interpolation and modulus errors.
%
% Synopsis
%   [interpolationError, modulusError] = errors(b, a, w, caseK)
%
% Description
%   errors compute the normalized L2 norm of the interpolation error
%   w-b(z)/a(z) for m distinct equidistant nodes in the unit circle z,
%   and
%   the L2 norm of th modulus condition error |a(x)|^2-|b(x)|^2-1 for 2m
%   -1
%   distinct equidistant nodes in the unit circle x.
%
% Inputs ([] are optional)
%   (vector) b: [m 1] vector representing the coefficients of the
%   numerator
%                 of a rational polynomial function of order n = m-1,
%                 b = [b0, ..., bn]^T -> b0 + b1*z^(-1) + ... + bn*z^(n).
%   (vector) a: [m 1] vector representing the coefficients of the
%                 denominator of a rational function of order n = m-1,
%                 a = [a0, ..., an]^T -> a0 + a1*z^(-1) + ... an*z^(-n).
%   (vector) [w]: [m 1] vector of interpolation values the function
%   should
%                   take at m distinct equidistant nodes on the unit circle
%   .
%   (string) [caseK]: string indicating the condition the function
%                     satisfies, can be:
%            - 'Defocusing': default case. k = -1 for the modulus
   condition
%                            error.
%            - 'Focusing': k = 1 for the modulus condition error.
%
% Outputs
%   (scalar) inteporlationError: normalized L2 norm of interpolation
   error,
%                                || w - b(z)/a(z) || / || w ||.
%   (scalar) modulusError: L2 norm of mudulus condition error,
%                          || |a(z)|^2 + k|b(z)|^2 - 1 ||.
%
% Examples
%   [interpError, modError] = errors([1 0.3], [1 0.4], [0.4 0.3])
%
% Author
%   Julian Uribe Jaramillo

%% Organize inputs and set parameters
if (nargin < 4)
    caseK = 'defocusing'; % Default
end

b = b(:);
a = a(:);
w = w(:);
caseK = lower(caseK);
m = length(b); % Number of interpolation points

switch (caseK)
    case 'focusing'
        k = 1;

    case 'defocusing'
        k = -1;

    otherwise
        error('caseK should be "Defocusing" or "Focusing"')
end

%% Compute point-value representations
B = fft(b);
A = fft(a);

%% Compute oversampled point-value representations
B2m = fft(b, 2*m-1);
A2m = fft(a, 2*m-1);

%% Compute errors
interpolationError = norm(B-w.*A)/norm(w);
modulusError = norm(abs(A2m).^2+k*abs(B2m).^2-1);

```

```
end
```

C-2 Focusing Solver

```
%% Load test data
fn = 'focusing';
load(fn);

%% Set hyper-parameters
dt0 = 0.2;
dt1 = 0.01;

nM = size(Z,2);
for iM = 1:nM
    z = Z(:,iM);
    w = W(:,iM);

    [b, a, h] = solver(w, dt0, dt1, fn);
    [interpError(iM), magError(iM)] = errors(b, a, w, 1, fn);
end
```

```
function [b, a, historicalData] = solver(w, dt0, dt1, caseK)
% SOLVER Finds the solution of a boundary analytic interpolation problem.
%
% Synopsis
%   [b, a, historicalData] = solver(w, dt0, dt1, path)
%
% Description
%   solver computes the denominator a and the numerator b of a rational
%   polynomial function that satisfies the modulus condition
%   |a(x)|^2+k|b(x)|^2=1, for all |x|=1, and the interpolation condition
%   b(z)/a(z) = w, where z are distinct equidistant nodes in the unit
%   circle.
%
% Inputs ([] are optional)
%   (vector) [w]: [m 1] vector of interpolation values the function
    should
%                   take at m distinct equidistant nodes on the unit circle
    .
%   (scalar) dt0: initial steplength.
%   (scalar) dt1: final steplength.
%   (string) [path]: string indicating the case. Can be:
%             - 'Defocusing': k = -1.
%             - 'Focusing': default case. k = 1.
%
% Outputs
%   (vector) b: [m 1] vector representing the coefficients of the
    numerator
%                 of a rational polynomial function of order n = m-1,
%                 b = [b0, ..., bn]^T -> b0 + b1*z^(-1) + ... + bn*z^(n).
%   (vector) a: [m 1] vector representing the coefficients of the
%                 denominator of a rational function of order n = m-1,
%                a = [a0, ..., an]^T -> a0 + a1*z^(-1) + ... an*z^(-n).
%   (struct) historicalData: data along the path. Contains:
%             - (scalar) .iter: number of predictor-corrector iterations.
%             - (vector) .interpolationError: [1 iter] vector of
   normalized
%                                          L2 interpolation errors at
%                                          each iteration.
%             - (vector) .modulusError: [1 iter] vector of L2 modulus
%                                    condition errors at each iteration
   .
%             - (vector) .b: [m iter] vector of b at each iteration.
%             - (vector) .a: [m iter] vector of a at each iteration.
%
% Examples
%   [b, a, historicalData] = solver([0.5 0.3], 0.1, 0.01, 'focusing')
%
% Author
%   Julian Uribe Jaramillo

%% Organize inputs and set parameters
if (nargin < 4)
    caseK = 'focusing';
end

w = w(:);
caseK = lower(caseK);
m = length(w); % Number of interpolation points
corIter = 0; % Number of corrector iterations

maxErrorPath = 1e-2; % Maximum interpolation error along the path

%% Compute number of iterations
i = 0;
t = 0;
while (t < 1)
    i = i + 1;

    %% Update arclength (t)
    dt = dt0+t*(dt1-dt0); % Steplength (\alpha)

    if (t + dt > 1)
        t = 1; % Arclength (t)
    else
        t = t+dt;
    end
end
iter = i;

%% Initialization
[b, a, c0, c1, dC] = initialize(w, caseK); % Initial approximation

% Set size of historical data
interpolationError = zeros(1, iter+1); % Interpolation error (e1(t))
```

```
modulusError = zeros(1, iter+1); % Modulus error (e2(t))
bH = zeros(m, iter+1); % Historical numerator coefficients (b(z,t))
aH = zeros(m, iter+1); % Historical denominator coefficients (a(z,t))

%% Compute historical data for initial approximation
[interpolationError(1), modulusError(1)] = errors(b, a, w, 1, caseK);
bH(:, 1) = b;
aH(:, 1) = a;

%% Iterate until end of arclength (t=1)
i = 1;
t = 0;
ct = c0;
while (t < 1)
    i = i+1;
    %     t
    %% Update the trajectory and find the local interpolation values
    [t, dt] = updateArclength(t, dt0, dt1);

    %% Predictor
    [b, a] = predictor(b, a, ct, dC, dt, caseK);
    ct = updatePath(c0, c1, t);

    %% Corrector
    [b, a] = corrector(b, a, ct, maxErrorPath, caseK);

    %% Save historical data
    [interpolationError(i), modulusError(i)] = errors(b, a, w, 1, caseK);
    bH(:, i) = b;
    aH(:, i) = a;
end

%% Organize outputs
b = b(:);
a = a(:);
historicalData.iter = iter;
historicalData.corIter = corIter;
historicalData.interpolationError = interpolationError;
historicalData.modulusError = modulusError;
historicalData.b = bH;
historicalData.a = aH;

%% Subfuctions
    function [b, a, c0, c1, dC] = initialize(w, caseK)
        %INITIALIZE Finds the initial polynomials

        switch (caseK)
            case 'focusing'
                k = 1;
            case 'defocusing'
                k = -1;
        end

        A = 1./(sqrt(1+k*abs(w).^2));
        a = ifft(A);

        b = a2b(a, 'interpolation', w);

        cc = xcorr(a)+k*xcorr(b);

        c0 = spec(cc);
        c0 = conj(flipud(c0(:)));

        c1 = round(c0);
        dC = c1-c0;

    end

    function [t, dt] = updateArclength(t, dt0, dt1)
        % UPDATEARCLENGTH Updates the arclength t

        %% Update the steplength
        dt = dt0+t*(dt1-dt0); % Steplength (\alpha)

        %% Update the arclength and keep it in range [0, 1]
        if (t + dt > 1)
            t = 1; % Arclength (t)
        else
            t = t+dt;
        end
    end

    function ct = updatePath(c0, c1, t)
        % UPDATEPATH Computes the local interpolation values in the path

        %% Update path
        ct = c0+t*(c1-c0); % Local interpolation values (w(t))

    end

    function [b, a] = predictor(b, a, ct, dC, dt, caseK)
        % PREDICTOR Approximates the next solution

        switch (caseK)
            case 'focusing'
                k = 1;
            otherwise
                k = -1;
        end

        A = fft(a);
        B = fft(b);
        C = fft(ct);
        DC = fft(dC);

        dA = (conj(C).*DC)./(conj(A) + k*conj(B).*w);
        A = A + dt*dA;

        a = ifft(A);

        b = a2b(a, 'interpolation', w);

    end

    function [b, a] = corrector(b, a, ct, maxError, caseK)
        % CORRECTOR Finds polynomials in the path close to the current
        % solution
        switch (caseK)
            case 'focusing'
                k = 1;
            otherwise
                k = -1;
        end

        cc0Local = xcorr(a) + k*xcorr(b);
        c0Local = spec(cc0Local);
        c0Local = conj(flipud(c0Local(:)));

        c1Local = ct;

        dCLocal = c1Local - c0Local;

        A = fft(a);
        B = fft(b);
        C = fft(c0Local);
        DCLocal = fft(dCLocal);

        err = norm(DCLocal);
        for K = 1:1000
            corIter = corIter + 1;
            aOld = a;
            bOld = b;

            dA = (conj(C).*DCLocal)./(conj(A) + conj(B).*w);
            A = A + dA;

            a = ifft(A);
            b = a2b(a, 'interpolation', w);

            cc0Local = xcorr(a) + k*xcorr(b);
            c0Local = spec(cc0Local);
            c0Local = conj(flipud(c0Local(:)));

            c1Local = ct;

            dCLocal = c1Local - c0Local;

            A = fft(a);
            B = fft(b);
```

```matlab
            C = fft(c0Local);
            DCLocal = fft(dCLocal);

            % Exit corrector if error improves less than machine
               precision
            if (norm(DCLocal) - err > eps)
                a = aOld;
                b = bOld;
                break;
            else
                err = norm(DCLocal);
            end
        end
    end
end
```

```matlab
function b = a2b(a, method, w)
%  a2b  Computes the numerator of a rational polynomial function.
%
% Synopsis
%   b = a2b(a, method, w)
%
% Description
%   a2b computes the numerator b from the denominator a of a rational
%   polynomial function. The polynomials satisfy either the modulus
%   condition |a(z)|^2-|b(z)|^2=1 for all |z|=1; or the interpolation
%   condition b(z_k)/a(z_k) = w_k.
%
% Inputs ([] are optional)
%   (vector) a [m 1] vector representing the coefficients of the
%              denominator of a rational polynomial function,
%              a = [a0, ..., an]^T -> a0 + a1*z^(-1) + ... an*z^(-n)
%   (string) [method] string indicating the condition the function
%                     satisfies, can be:
%                      - 'Modulus': the polynomials satisfy
%                        |a(z)|^2-|b(z)|^2=1 for all |z|=1.
%                      - 'Interpolation': the polynomials satisfy
%                        b(z)/a(z) = w, where z are m points in the unit
%                        circle and w are given interpolation values.
%   (vector) [w] [m 1] vector representing the values the function
%                should take at m points on the unit circle.
%
% Outputs
%   (vector) b [m 1] vector representing the coefficients of the
   numerator
%              of a rational polynomial function,
%              b = [b0, ..., bn]^T -> b0 + b1*z^(-1) + ... bn*z^(-n)
%
% Examples
%   b = a2b([1 0.3].', 'Interpolation', [0.4 0.3].')
%
% Author
%   Julian Uribe Jaramillo
```

```

a = a(:);

method = lower(method);
m = length(a);

if (nargin < 2)
    method = 'modulus'; % Default
end

switch (method)
    case 'defocusing'
        aa = xcorr(a);

        bb = aa;
        bb(m) = bb(m) - 1;

        b = spec(bb);
        b = b(:);

    case 'focusing'
        aa = xcorr(a);

        bb = -aa;
        bb(m) = 1 - bb(m);

        b = spec(bb);
        b = b(:);

    case 'interpolation'
        if (nargin < 3)
            error('Interpolation values missing')
        else
            A = fft(a);
            B = w.*A;
            b = ifft(B);
        end

    otherwise
        error('Method should be "Modulus" or "Interpolation"')
end
end
```

```
function [interpolationError, modulusError] = errors(b, a, w, c, caseK)
% ERRORS Compute the interpolation and modulus errors.
%
% Synopsis
%   [interpolationError, modulusError] = errors(b, a, w)
%
% Description
%   errors compute the normalized L2 norm of the interpolation error
%   w-b(z)/a(z) for m distinct equidistant nodes in the unit circle z,
    and
```

```
%   the L2 norm of th modulus condition error |a(x)|^2-|b(x)|^2-1 for 2m
    -1
%   distinct equidistant nodes in the unit circle x.
%
% Inputs ([] are optional)
%   (vector) b: [m 1] vector representing the coefficients of the
    numerator
%               of a rational polynomial function of order n = m-1,
%               b = [b0, ..., bn]^T -> b0 + b1*z^(-1) + ... + bn*z^(n).
%   (vector) a: [m 1] vector representing the coefficients of the
%               denominator of a rational function of order n = m-1,
%               a = [a0, ..., an]^T -> a0 + a1*z^(-1) + ... an*z^(-n).
%   (vector) [w]: [m 1] vector of interpolation values the function
    should
%                 take at m distinct equidistant nodes on the unit circle
    .
%   (string) [caseK]: string indicating the condition the function
%                     satisfies, can be:
%             - 'Defocusing': default case. k = -1 for the modulus
    condition
%                             error.
%             - 'Focusing': k = 1 for the modulus condition error.
%
% Outputs
%   (scalar) inteporlationError: normalized L2 norm of interpolation
    error,
%                                || w - b(z)/a(z) || / || w ||.
%   (scalar) modulusError: L2 norm of mudulus condition error,
%                          || |a(z)|^2 + k|b(z)|^2 - 1 ||.
%
% Examples
%   [interpError, modError] = errors([1 0.3], [1 0.4], [0.4 0.3])
%
% Author
%   Julian Uribe Jaramillo

%% Organize inputs and set parameters
m = length(b); % Number of interpolation points

if (nargin < 5)
    caseK = 'defocusing'; % Default
end

if (nargin < 4)
    caseK = 'defocusing'; % Default
    c = 1;
end

c = c(:);
b = b(:);
a = a(:);
w = w(:);
caseK = lower(caseK);
```

```

switch (caseK)
    case 'focusing'
        k = 1;

    case 'defocusing'
        k = -1;

    otherwise
        error('caseK should be "Defocusing" or "Focusing"')
end

%% Compute point-value representations
B = fft(b);
A = fft(a);

%% Compute oversampled point-value representations
B2m = fft(b, 2*m-1);
A2m = fft(a, 2*m-1);
C2m = fft(c, 2*m-1);
C2m = C2m(:);

%% Compute errors
interpolationError = norm(w-B./A)/norm(w);
modulusError = norm(abs(A2m).^2+k*abs(B2m).^2-abs(C2m).^2);

end
```

C-3 Spectral Factorization

```
N = 1:12;
N = 2.^N-1;
nN = length(N);

m = 100;

for iN = 1:nN
    n = N(iN);

    f0 = [(1/2)*ones(1, n) 1];
    f0 = f0(:);
    u0 = [2 ones(1, n)];
    u0 = u0(:);

    pp = conv(f0, u0);

    pp = pp(:);

    s = wienerHopf(pp, m);

    errWH(iN) = norm(s-f0);

    s = wilsonBurg(pp, m, f0);

    errWB(iN) = norm(s-f0);

end
```

```
function s = wienerHopf(pp, m)
% WIENERHOPF Polynomial spectral factorization by Wiener-Hopf algorithm.
%
% Synopsis
%   s = wienerHopf(pp, m)
%
% Description
%   wilsonBurg computes the spectral factorization s of a Laurent
%   polynomial pp.
%
% Inputs ([] are optional)
%   (vector) pp:  [2n-1 1] vector of coefficients of the Laurent
   polynomial
%                  pp(z) = pp(1)*z^(-n)+...+pp(2*n+1)*z^n.
%   (scalar) [m]: maximum number of iterations.
%
% Outputs
%   (vector) s: [n 1] vector of coefficients of a polynomial
%                s(z) = s(1)z^(-n)+...+s(n), with all the roots of s(z) >
   1
%                and pp(z) = s(z)*conj(s(z)).
%
% Examples
%   s = wienerHopf([1 2.5 1], 100)
%
% Author
%   Julian Uribe Jaramillo

pp = pp(:);

n = length(pp);

if isempty(m)
    m = 32*n;
end

nP = floor((n-1)/2);
nM = ceil((n-1)/2);

pp0 = pp(1:nP);
pp1 = pp((nP+1):end);

Tb = toeplitz(pp0, [pp0(1) zeros(1, nP-1)]);
Ta = toeplitz(pp1(1:nP), [pp1(1); pp0(end:-1:end-nP+2)]);

x = inv(Ta);
x = x(:,1);
```

```

f = Tb*x;

for i = 1:m
    s = [f; 1];

    ss = xcorr(s);
    s0 = pp(nM+1)/ss(nM+1);

    s = sqrt(s0)*s;
    s = s(:);

    F0 = toeplitz(f, [f(1) zeros(1, nM)]);

    F1 = toeplitz([1; zeros(nM, 1)], [1 fliplr(f.') zeros(1, nM-nP)]);

    u = F1\pp1;

    e = pp0 - F0*u;

    G0 = F0(:, 1:(nM-nP+1));
    G1 = F1(1:(nM-nP+1), 1:(nM-nP+1));

    u0 = u(1:nP);
    u1 = u(nP+1:end);

    q = G1\u1;
    r = u0 - G0*q;

    de = zeros(length(r));
    de(:,1) = r;
    for j = 1:(nP-1)
        v = de(:,j);
        de(:,j+1) = [0; v(1:end-1)] - v(end)*f;
    end
    de = -de;

    fOld = f;
    f = f-de\e;

    if (norm(f-fOld) < eps)
        f = fOld;
        break;
    end
end

s = [f; 1];

% Uncomment to multiply polynomial by constant gain
% ss = xcorr(s);
% s0 = pp(nM+1)/ss(nM+1);
%
% s = sqrt(s0)*s;
% s = s(:);

end
```

```
function s = wilsonBurg(pp, m)
% WILSONBURG Polynomial spectral factorization by Wilson-Burg algorithm.
%
% Synopsis
%   s = wilsonBurg(pp, m)
%
% Description
%   wilsonBurg computes the spectral factorization s of a Laurent
%   polynomial pp.
%
% Inputs ([] are optional)
%   (vector) pp:  [2n-1 1] vector of coefficients of the Laurent
    polynomial
%                  pp(z) = pp(1)*z^(-n)+...+pp(2*n+1)*z^n.
%   (scalar) [m]: maximum number of iterations.
%
% Outputs
%   (vector) s: [n 1] vector of coefficients of a polynomial
%                s(z) = s(1)z^(-n)+...+s(n), with all the roots of s(z) >
    1
%                and pp(z) = s(z)*conj(s(z)).
%
% Examples
%   s = wilsonBurg([1 2.5 1], 100)
%
% Author
%   Julian Uribe Jaramillo

pp = pp(:);

n = length(pp);

if isempty(m)
    m = 32*n;
end

n = ceil(length(pp)/2);

% Initialize polynomials
y = zeros(n, 1);
y(1) = 1;

s = zeros(n, 1);
s(1) = 1;

for i = 1:m
    % Generate convolution and deconvolution matrix
    convMatrix = toeplitz(s, [s(1) zeros(1, n-1)]);

    % Generate convolution and deconvolution matrix
    deconvMatrix = toeplitz([s; zeros(n-1, 1)], [s(1) zeros(1, 2*n-2)]);

    % Compute yy(z) = ss(z)/a(z) by forward deconvolution
    yy = deconvMatrix\pp;

    % Compute yy(z) = ss(z)/(a*(1/z)a(z)) by adjoint deconvolution
    yy = deconvMatrix'\yy;

    % Compute y(z) = 1+yy(z), remove negative lags, and keep half zero
       lag
    y(2:n) = (yy(n+1:end)+conj(yy(n-1:-1:1)))/(2*yy(n));

    % Compute a(z) = y(z)*a(z);
    sOld = s;
    s = convMatrix*y;

    if (norm(s - sOld) < eps)
        break;
    end

end

% Uncomment to multiply polynomial by constant gain
% s = sqrt(yy(n))*s;
% s = conj(flipud(s));

end
```

Bibliography

[1] E. Agrell, M. Karlsson, A. R. Chraplyvy, D. J. Richardson, P. M. Krummrich, P. Winzer, K. Roberts, J. K. Fischer, F. R. Kschischang, A. Lord, J. Prat, I. Tomkos, J. E. Bowers, S. Srinivasan, M. Brandt-Pearce, and N. Gisin, "Roadmap of optical communications," *Journal of optics*, vol. 18, no. 6, p. 063 002, 2016.

[2] A. Splett, C. Kurtzke, and K. Petermann, "Ultimate transmission capacity of amplified optical fiber communication systems taking into account fiber nonlinearities," vol. 2, 1993, pp. 41–44.

[3] M. J. Ablowitz and A. D. Trubatch, "Integrable nonlinear Schrödinger systems and their soliton dynamics," *Dynamics of Partial Differential Equations*, vol. 1, no. 3, pp. 239–299, 2004.

[4] M. J. Ablowitz, D. J. Kaup, A. C. Newell, and H. Sugar, "The inverse scattering transform-Fourier analysis for nonlinear problems," *Studies in Applied Mathematics*, vol. 53, no. 4, pp. 249–315, 1974.

[5] T. Trogdon and S. Olver, "Numerical inverse scattering for the focusing and defocusing nonlinear Schrödinger equations," 2149, vol. 469, 2013, p. 20 120 330.

[6] D. Lavery, R. Maher, D. S. Millar, B. C. Thomsen, P. Bayvel, and S. J. Savory, "Digital coherent receivers for long-reach optical access networks," *Journal of Lightwave Technology*, vol. 31, no. 4, pp. 609–620, 2013.

[7] S. Wahls and V. Vaibhav, "Fast inverse nonlinear Fourier transforms for continuous spectra of Zakharov-Shabat type," *ArXiv e-prints*, 2016. arXiv: `1607.01305`. [Online]. Available: `http://adsabs.harvard.edu/abs/2016arXiv160701305W`.

[8] S. Wahls and H. V. Poor, "Inverse nonlinear Fourier transform via interpolation: The Ablowitz-Ladik case," 2014, pp. 1848–1855.

[9] M. J. Ablowitz and J. F. Ladik, "Nonlinear differential-difference equations and Fourier analysis," *Journal of Mathematical Physics*, vol. 17, no. 6, pp. 1011–1018, 1976.

[10] C. I. Byrnes, T. T. Georgiou, and A. Lindquist, "A generalized entropy criterion for Nevanlinna-Pick interpolation with degree constraint," *IEEE Transactions on Automatic Control*, vol. 46, no. 6, pp. 822–839, 2001.

[11] L. Baratchart, M. Olivi, and F. Seyfert, "Boundary Nevanlinna-Pick interpolation with prescribed peak points. application to impedance matching," *HAL e-prints*, 2016. hal: 01377782. [Online]. Available: https://hal.inria.fr/hal-01377782.

[12] R. Nagamune, "A robust solver using a continuation method for Nevanlinna-Pick interpolation with degree constraint," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 113–117, 2003.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT Press, 2009.

[14] S. G. Krantz, *Handbook of complex variables*. Birkhauser, 1999.

[15] A. Tannenbaum, "Feedback stabilization of linear dynamical plants with uncertain in the gain factor," *International Journal of Control*, vol. 32, no. 1, pp. 1–16, 1980.

[16] ——, "Modified Nevanlinna-Pick interpolation and feedback stabilization of linear plants with uncertainty in the gain factor," *International Journal of Control*, vol. 36, no. 2, pp. 331–336, 1982.

[17] J. A. Ball and S. T. Host, "Robust control, multidimensional systems and multivariable Nevanlinna-Pick interpolation," *ArXiv e-prints*, 2009. arXiv: 0906.3363. [Online]. Available: http://adsabs.harvard.edu/abs/2009arXiv0906.3363B.

[18] P. Delsarte, Y. Genin, and Y. Kamp, "On the role of the Nevanlinna-Pick problem in circuit and system theory," *Circuit Theory and Applications*, vol. 9, no. 2, pp. 177–187, 1981.

[19] C. I. Byrnes, T. T. Georgiou, and A. Lindquist, "A new approach to spectral estimation: A tunable high-resolution spectral estimator," *IEEE Transactions on Signal Processing*, vol. 48, no. 11, pp. 3189–3205, 2000.

[20] J. Karlsson, "Inverse problems in analytic interpolation for robust control and spectral estimation," PhD thesis, Royal institute of technology (KTH), 2008.

[21] J. Doyle, B. Francis, and A. Tannenbaum, *Feedback control theory*. Macmillan Publishers, 1990.

[22] R. Nevanlinna, "Über beschränkte analytische funktionen," *Annales Academiae Scientiarum Fennicae Series A 32*, 1929.

[23] G. Pick, "Über die beschränkungen analytischer funktionen, welche durch vorgegebene funktionswerte bewirkt werden," *Mathematische Annalen*, vol. 77, no. 1, pp. 7–23, 1915.

[24] R. Nagamune, "Robust control with complexity constraint: A nevanlinna-pick interpolation approach," PhD thesis, Royal institute of technology (KTH), 2002.

[25] S. T. Le, J. E. Prilepsky, P. Rosa, J. D. Ania-Castañón, and S. K. Turitsyn, "Nonlinear inverse synthesis for optical links with distributed raman amplification," *Journal of Lightwave Technology*, vol. 34, no. 8, pp. 1778–1786, 2016.

[26] Karigiannis, "The inverse scattering transform and the integrability of nonlinear evolution equations," Master's thesis, Harvard University, 1998.

[27] S. T. Le, J. E. Prilepsky, and S. K. Turitsyn, "Nonlinear inverse synthesis technique for optical links with lumped amplification," *Optics Express*, vol. 23, no. 7, pp. 8317–8328, 2015.

[28] E. L. Allgower and K. Georg, *Introduction to numerical continuation methods*. Society for Industrial and Applied Mathematics, 2003.

[29] B. Dumitrescu, *Positive trigonometric polynomials and signal processing applications*. Springer, 2007.

[30] R. Seydel, *Practical bifurcation and stability analysis*. Springer, 2010.

[31] S. Ponnusamy and H. Silverman, *Complex variables with applications*. Birkhäuser, 2006.

[32] O. V. Belai, L. L. Frumin, E. V. Podivilov, and D. A. Shapiro, "Efficient numerical method of the fiber bragg grating synthesis," *Journal of the Optical Society of America B*, vol. 24, no. 7, pp. 1451–1457, 2007.

[33] H. Boche and V. Pohl, "Spectral factorization, whitening- and estimation filter – stability, smoothness properties and fir approximation behavior," *ArXiv e-prints*, 2005. arXiv: cs/05080183. [Online]. Available: https://arxiv.org/abs/cs/0508018v1.

[34] M. Verhaegen and V. Verdult, *Filtering and System Identification*. Cambridge University Press, 2007.

[35] S. Fomel, P. Sava, J. Rickett, and J. F. Claerbout, "The Wilson-Burg method of spectral factorization with application to helical filtering," *Geophysical Prospecting*, vol. 51, pp. 409–420, 2003.

[36] A. Böttcher and M. Halwass, "Wiener–Hopf and spectral factorization of real polynomials by Newton's method," *Linear Algebra and its Applications*, vol. 438, pp. 4760–4805, 2013.

[37] A. H. Sayed and T. Kailath, "A survey of spectral factorization methods," *Numerical Linear Algebra with Applications*, vol. 8, pp. 177–187, 2001.