Technical University of Denmark

Numerical Analysis of Gabor Frame Sets

Bachelor's Thesis in Mathematics and Technology

Authors
Clara Hollenbeck (s214722)
Jakob Cetti Hansen (s214695)
Hans Harck Tønning (s214700)

Supervisors
Jakob Lemvig
Marzieh Hasannasabjaldehbakhani

June 16, 2024

Abstract

One of the main problems in Gabor analysis is finding parameters α and β for a window function $\phi \in L^2(\mathbb{R})$ such that $\{e^{2\pi i m\beta x}\phi(x-k\alpha) : k, m \in \mathbb{Z}\}$ forms a frame with the frame bounds A and B. We correct the method for computing frame bounds presented in [13], and give an overview of different methods to compute the frame set and frame bounds of window functions in a subspace of $L^2(\mathbb{R})$. We develop a toolbox for estimating the frame set and the frame bounds numerically in the programming language Julia. Furthermore, we have studied the B-splines of order $m \geq 2$ and showed that for any positive, non-integer β it is possible to find a corresponding α such that (α, β) is in the frame set of the B-spline.

Contents

Chapter 1

Introduction

Gabor frames are used in a wide variety of scientific fields, especially in those concerning time-frequency analysis e.g. to analyze music signals [9], and to reconstruct signals [10], among other things. One of the main problems in Gabor analysis is finding parameters α and β for a window function $\phi \in L^2(\mathbb{R})$ such that $\{e^{2\pi i m\beta x}\phi(x-k\alpha) : k, m \in \mathbb{Z}\}$ forms a frame. That is to say, there exists positive frame bounds A and B such that

$$A\left\|f\right\|^2 \leq \sum_{m,n\in\mathbb{Z}} |\langle f, e^{2\pi i m\beta x}\phi(x-k\alpha)\rangle|^2 \leq B\left\|f\right\|^2, \qquad f \in L^2(\mathbb{R}).$$

This problem has shown to be difficult to solve and the full frame set has only been found for a small set of functions, notably the Gaussian, the characteristic function, and the one sided and two sided exponential functions [17].

In this thesis, we use numerical analysis to estimate the frame set and frame bounds of functions in a special subspace of $L^2(\mathbb{R})$. This method is based on the work by Ghosh and Selvan in [13], but with a small correction in the way the frame bounds are computed, as the method they proposed sometimes gives lower frame bounds that were larger than their corresponding upper frame bounds. We mostly focus on studying the frame set for B-splines of order $m \geq 2$ and we show that for any positive, non-integer β it is possible to find a corresponding α such that (α, β) is in the frame set of the B-spline.

We start by presenting some known definitions and results from analysis, which are used throughout this thesis. We then give a short introduction to frames, and some of their properties, before we define Gabor frames. Afterward, we explain the Zibulski-Zeevi method for finding frame bounds and prove a corrected variant of the method proposed by Ghosh and Selvan.

We then present several methods that allow simpler computation of a subset of the frame set shown by Ghosh and Selvan. Afterward, we present our toolbox and explain how to use it and how it works. Finally, we state some results for the frame set of B-splines from our numerical analysis, as well as discuss the different pros and cons of each of the methods. As a small note at the end we show that these methods also work on other classes of functions than the B-splines.

Chapter 2

Preliminaries

In this chapter, we will give some necessary definitions, lemmas, and theorems, mostly from the book "Functions, Spaces, and Expansions" by Ole Christensen [6] that are used throughout the thesis. It is however assumed that the reader is familiar with basic analysis. The theorems and lemmas are mostly given without proofs but with references to where proofs can be found.

2.1 Unitary and projection operators

Let $\mathcal{H}$ with $\mathcal{H} \neq \{0\}$ denote a separable Hilbert space.

Definition 2.1.1 (Self-adjoint and unitary operator) *[6, p. 74] Let $T : \mathcal{H} \to \mathcal{H}$ be a bounded linear operator.*
(i) T is ***self-adjoint*** *if $T = T^*$*
(ii) T is ***unitary*** *if $TT^* = T^*T = I$.*

We see from the definition that if T is unitary, then it is a bijection since it has an inverse $T^{-1} = T^*$. Observe also that if T is unitary then $\langle Tf, Tg\rangle = \langle T^*f, T^*g\rangle = \langle f, g\rangle$ and $\|Tf\| = \|T^*f\| = \|f\|$. This property makes the unitary operators very special when working with Gabor frames as we will see later.

We also define orthogonal projections for Hilbert spaces.

Definition 2.1.2 (Orthogonal projections) *[6, pp. 67, 74] Let V be a closed subspace of a Hilbert space $\mathcal{H}$. We define the* ***orthogonal complement*** *$V^\perp$ of the subspace V of $\mathcal{H}$ as*

$$V^\perp = \{w \in \mathcal{H} \mid \langle v, w\rangle = 0, \ \forall v \in V\}.$$

Furthermore, $v \in \mathcal{H}$ can be written as $v = v_1 + v_2$, where $v_1 \in V$ and $v_2 \in V^\perp$ are unique. The ***orthogonal projection*** *P onto V is defined by*

$$Pv = v_1.$$

The following lemma is useful when working with orthogonal projections.

Lemma 2.1.3 *[6, p. 74] Let $V \neq \{0\}$ be a closed subspace of a Hilbert space $\mathcal{H}$. Then the orthogonal projection of P of $\mathcal{H}$ onto V has the following properties:*

(i) P is linear and bounded, with $\|P\| = 1$
(ii) P is self-adjoint
(iii) $P^2 = P$.

The following are some very important linear unitary operators from $L^2(\mathbb{R})$ to $L^2(\mathbb{R})$. They will be used extensively throughout this thesis.

Definition 2.1.4 (Translation, modulation, dilation) *[6, p. 120] Let $a, b, c \in \mathbb{R}$ such that $c > 0$ and $f \in L^2(\mathbb{R})$. Then the operators **translation** $T_a : L^2(\mathbb{R}) \to L^2(\mathbb{R})$, **modulation** $E_b : L^2(\mathbb{R}) \to L^2(\mathbb{R})$, and **dilation** $C_c : L^2(\mathbb{R}) \to L^2(\mathbb{R})$ are defined by:*
(i) $(T_a f)(x) := f(x - a)$
(ii) $(E_b f)(x) := e^{2\pi i b x} f(x)$
(iii) $(D_c f)(x) := \frac{1}{\sqrt{c}} f(\frac{x}{c})$.

Since these operators will be used so frequently, we will state some properties of them in the following theorem and lemma.

Theorem 2.1.5 *The operators T_a, E_b, and D_c are unitary linear operators and the following relations hold:*
(i) $T_a^{-1} = T_{-a} = T_a^*$
(ii) $E_b^{-1} = E_{-b} = E_b^*$
(iii) $D_c^{-1} = D_{1/c} = D_c^*$.

The three operators can also be composed with each other, which is used to define the Gabor frames later on.

Lemma 2.1.6 *[6, p. 123] For all $a, b, c \in \mathbb{R}$ such that $c > 0$, it holds that:*
(i) $T_a E_b = e^{-2\pi i b a} E_b T_a$
(ii) $T_a D_c = D_c T_{a/c}$
(iii) $D_c E_b = E_{b/c} D_c$.

2.2 Fourier Transform

The Fourier transform is one of the most essential tools in engineering. It is especially important in signal processing, as the Fourier transform shows which frequencies are present in a given signal. In this section we define the Fourier transform, and some of its properties, along with the commutation relations for T_a, E_b, and D_c defined in the section above.

Definition 2.2.1 (The Fourier Transform) *[6, p. 136] For $f \in L^1(\mathbb{R})$, the **Fourier transform** $\widehat{f} : \mathbb{R} \to \mathbb{C}$ is defined by:*

$$\widehat{f}(\gamma) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \gamma} dx, \quad \gamma \in \mathbb{R}.$$

We will denote the Fourier transform of f by $\mathcal{F}f$.

Since we will work with functions in $L^2(\mathbb{R})$, we will use the following extension of the Fourier transform to $L^2(\mathbb{R})$ and the fact that it becomes a unitary operator:

Theorem 2.2.2 *[6, p. 143] The Fourier transform can be extended to a unitary mapping of $L^2(\mathbb{R})$ onto $L^2(\mathbb{R})$. Furthermore, it holds for $f \in L^2(\mathbb{R})$ that*

$$\hat{\hat{f}}(x) = f(-x), \quad x \in \mathbb{R}.$$

The Fourier transform also commutes with the translation, modulation, and dilation operators as described in the following theorem.

Theorem 2.2.3 *[6, p. 137] For all $a, b, c \in \mathbb{R} : c > 0$ it holds that*
(i) $\mathcal{F}T_a = E_{-a}\mathcal{F}$
(ii) $\mathcal{F}E_b = T_b\mathcal{F}$
(iii) $\mathcal{F}D_c = D_{1/c}\mathcal{F}$.

Furthermore, we show how differentiation and the Fourier transform commute.

Theorem 2.2.4 *[6, p. 137] If $f \in L^1(\mathbb{R})$ such that f is differentiable with $f' \in L^1(\mathbb{R})$ and $f(x) \to 0$ as $x \to \pm\infty$ then it holds that*

$$(\mathcal{F}f')(\gamma) = 2\pi i\gamma\widehat{f}(\gamma), \quad \gamma \in \mathbb{R}.$$

Lastly, we define the convolution of two functions f and g.

Definition 2.2.5 (Convolution) *[6, p. 145] Let $f, g \in L^1(\mathbb{R})$ then the **convolution** $f * g : \mathbb{R} \to \mathbb{C}$ is defined by*

$$(f * g)(y) := \int_{-\infty}^{\infty} f(y - x)g(x)\, dx, \quad y \in \mathbb{R}.$$

We state some basic properties of convolutions and how the Fourier transform affects it.

Theorem 2.2.6 *[6, p. 147] If $f, g \in L^1(\mathbb{R})$ it holds that:*
(i) $f * g = g * f$
(ii) $\widehat{f * g} = \widehat{f} \cdot \widehat{g}$.

2.3 B-splines

In this section, we state the definition of B-splines and some of their properties. B-splines are a very nice class of functions as they are continuous, differentiable except at a finite number of points, and have compact support. Our thesis will mainly focus on finding the frame bounds and frame set of the B-splines.

Definition 2.3.1 (B-spline) *Let $m \in \mathbb{N}$. The **B-spline**, Q_m, of order m is defined as*

$$Q_1(x) = \chi_{[-\frac{1}{2},\frac{1}{2}]}(x) \quad \text{and} \quad Q_{m+1}(x) = (Q_m * Q_1)(x).$$

We also define the sinc-function

Definition 2.3.2 (Sinc) *The **sinc**-function is given by*

$$\operatorname{sinc}(x) := \begin{cases} \frac{\sin(\pi x)}{\pi x} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0. \end{cases}$$

The sinc-function is continuous since $\lim_{x \to 0} \frac{\sin(\pi x)}{\pi x} = 1$. Therefore, we will use $\frac{\sin(\pi x)}{\pi x}$ and $\operatorname{sinc}(x)$ interchangeably throughout this thesis. Furthermore, we easily see that sinc has roots at $\mathbb{Z} \setminus \{0\}$, since $\sin(\pi x) = 0$ for $x \in \mathbb{Z}$.

The B-splines have the following properties, and closed form expressions, which makes them easy to work with.

Proposition 2.3.3 (B-spline properties) *[13, p. 7] The B-spline Q_m belongs to $C_c^{m-2}(\mathbb{R})$ with compact support $[-\frac{m}{2}, \frac{m}{2}]$. The B-spline Q_m, its Fourier transform, and its derivatives can be computed with the following formulas:*
(i) $Q_m(x) = \frac{1}{(m-1)!} \sum_{j=0}^{m} (-1)^j \binom{m}{j} (x + \frac{m}{2} - j)_+^{m-1}$ *for* $m \geq 2$
(ii) $\widehat{Q}_m(w) = \left(\frac{\sin(\pi w)}{\pi w}\right)^m$ *for* $m \geq 1$
(iii) $Q_m'(x) = Q_{m-1}(x + \frac{1}{2}) - Q_{m+1}(x - \frac{1}{2})$ *for* $m > 2$
where $x_+ = \max(0, x)$.

Proof. The proofs of $Q_m \in C_c^{m-1}(\mathbb{R})$ with $\operatorname{supp} Q_m = [-\frac{m}{2}, \frac{m}{2}]$ and (i) can be seen in [6, pp. 205, 208, 246–248]. We will prove (ii) and (iii).

We first consider the Fourier transform of $Q_1(x)$:

$$\widehat{Q}_1(\gamma) = \int_{-\infty}^{\infty} \chi_{[-\frac{1}{2}, \frac{1}{2}]}(x) e^{-2\pi i x \gamma} \, dx = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-2\pi i x \gamma} \, dx = \frac{\sin(\pi \gamma)}{\pi \gamma}.$$

Then as $Q_m = Q_{m-1} * Q_1$, it now follows by Theorem 2.2.6, that

$$\widehat{Q}_m(\gamma) = \widehat{Q_{m-1} * Q_1} = \widehat{Q}_{m-1}(\gamma) \cdot \widehat{Q}_1(\gamma) = \widehat{Q}_1(\gamma)^m = \left(\frac{\sin(\pi \gamma)}{\pi \gamma}\right)^m.$$

which shows (ii). To show (iii) we first define

$$R(x) = Q_{m-1}\left(x + \frac{1}{2}\right) - Q_{m-1}\left(x - \frac{1}{2}\right) = T_{-\frac{1}{2}} Q_{m-1}(x) - T_{\frac{1}{2}} Q_{m-1}(x).$$

As the Fourier transform is unitary in $L^2(\mathbb{R})$ we just need to show (iii) in the Fourier domain. By using the commutation relations in Theorem 2.2.3 and Theorem 2.2.4 we see that

$$\begin{aligned} \widehat{R}(x) &= E_{\frac{1}{2}} \widehat{Q}_{m-1}(x) - E_{-\frac{1}{2}} \widehat{Q}_{m-1}(x) = (e^{\pi i x} - e^{-\pi i x}) \widehat{Q}_{m-1}(x) \\ &= 2i \sin(\pi x) \left(\frac{\sin(\pi x)}{\pi x}\right)^{m-1} = 2\pi x i \left(\frac{\sin(\pi x)}{\pi x}\right)^m = \widehat{Q}_m'(x). \end{aligned}$$

This completes the proof. □

The B-splines and their Fourier transform can be seen in the following figures.

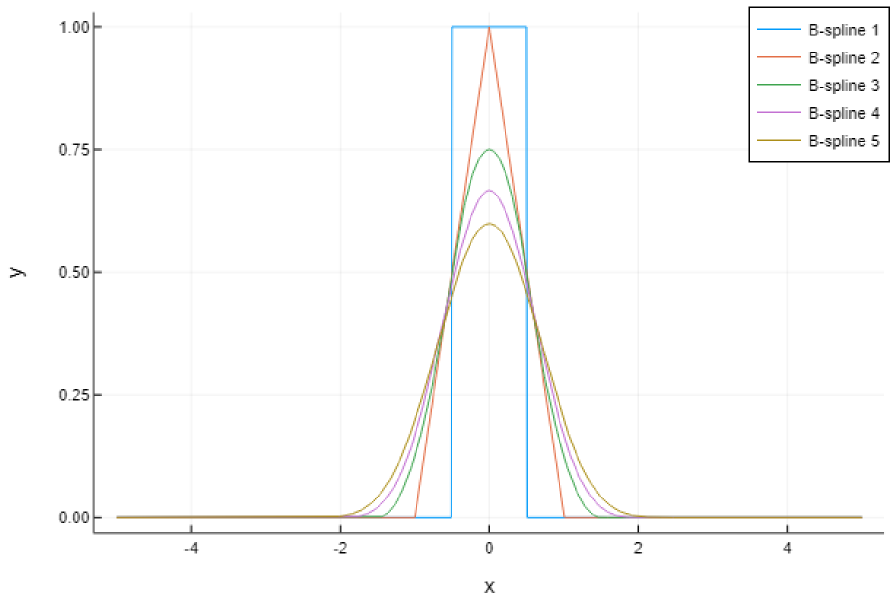

Figure 2.1: The B-splines Q_m for $m \in \{2, 3, 4, 5\}$.

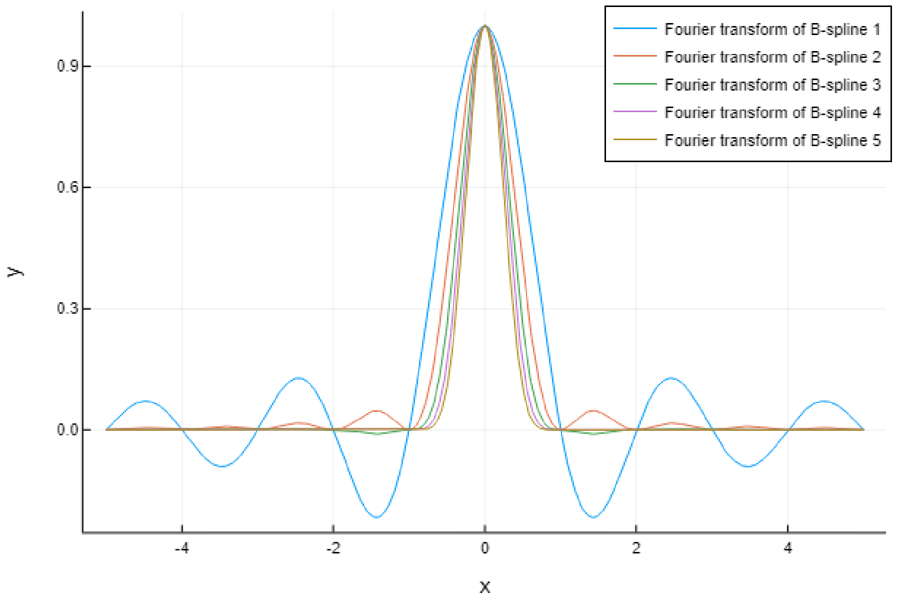

Figure 2.2: The Fourier transform of the B-splines $\widehat{Q}_m$ for $m \in \{2, 3, 4, 5\}$.

Chapter 3

Gabor Frames

In this chapter, we give some definitions and theorems about frames in general and Gabor frames in particular. A lot of the definitions, lemmas, and theorems are from the book "An Introduction to Frames and Riesz Bases" by Ole Christensen [4], which we refer to for a deeper diving study.

3.1 Frames

We start by defining frames as well as showing some general results for frames.

Definition 3.1.1 *[4, pp. 120–121] A sequence $\{f_k\}_{k\in\mathbb{Z}}$ of elements in $\mathcal{H}$ is a* ***frame*** *for $\mathcal{H}$ if there exists constants $A, B > 0$ such that*

$$A\left\|f\right\|^2 \leq \sum_{k\in\mathbb{Z}} |\langle f, f_k\rangle|^2 \leq B\left\|f\right\|^2, \quad f \in \mathcal{H}.$$

The constants A and B are called ***frame bounds****, and they are not unique. The optimal lower frame bound is the supremum over all lower frame bounds and the optimal upper frame bound is the infimum over all upper frame bounds.*

A frame is ***tight*** *if we can choose $A = B$ as frame bounds. A tight frame with bounds $A = B = 1$ is called a* ***Parseval frame****.*

An important feature of frames is the reconstruction formula.

Theorem 3.1.2 *[4, p. 119] Let $\{f_k\}_{k\in\mathbb{Z}}$ be a frame for $\mathcal{H}$. Then $f \in \mathcal{H}$ can be written as*

$$f = \sum_{k\in\mathbb{Z}} c_k f_k$$

where the coefficients c_k are unique.

This theorem means that frames act like bases, but without necessarily being linearly independent. Bases are also frames and if we add linear dependent vectors to a basis, then it would still be a frame even though it is no longer a basis. The redundancy in frames can add to more stability and noise reduction when reconstructing signals, and frames can thus be more preferable to use instead of bases [2]. Different reconstruction algorithms have been proposed in [10] and [2].

We often also look at sequences $\{f_k\}_{k\in\mathbb{Z}}$ of elements in $\mathcal{H}$ that do not form a frame for all of $\mathcal{H}$, which leads to the following definition.

Definition 3.1.3 *[4, p. 121] Let $\{f_k\}_{k\in\mathbb{Z}}$ be sequence of elements in $\mathcal{H}$. We say that $\{f_k\}_{k\in\mathbb{Z}}$ is a frame sequence if it is a frame for $\overline{\text{span}}\{f_k\}_{k\in\mathbb{Z}}$.*

As we mainly focus on computing the frame bounds, we want to know how changes to a frame affect the frame bounds. We present the following useful lemmas about what happens with the frame bounds when we manipulate a frame $\{f_k\}_{k\in\mathbb{Z}}$.

Lemma 3.1.4 *Let $c \in \mathbb{C} \setminus \{0\}$, then $\{f_k\}_{k\in\mathbb{Z}}$ is a frame with frame bounds A and B if and only if $\{cf_k\}_{k\in\mathbb{Z}}$ is a frame with frame bounds $|c|^2A$ and $|c|^2B$.*

Proof. Recall from Definition 3.1.1 that:

$$A\left\|f\right\|^2 \leq \sum_{k\in\mathbb{Z}} |\langle f, f_k\rangle|^2 \leq B\left\|f\right\|^2, \quad f \in \mathcal{H}.$$

By multiplying with $|c|^2$ and using that $|c|^2|\langle f, f_k\rangle|^2 = |\overline{c}\langle f, f_k\rangle|^2 = |\langle f, cf_k\rangle|^2$, it follows that

$$|c|^2A\left\|f\right\|^2 \leq \sum_{k\in\mathbb{Z}} |\langle f, cf_k\rangle|^2 \leq |c|^2B\left\|f\right\|^2, \quad f \in \mathcal{H}.$$

This shows that $\{cf_k\}_{k\in\mathbb{Z}}$ is a frame with frame bounds $|c|^2A$ and $|c|^2B$. □

Lemma 3.1.5 *Let $U : \mathcal{H} \to \mathcal{H}$ be a unitary operator and $\{c_k\}_{k\in\mathbb{Z}}$ be a sequence of constants such that $|c_k| = 1$ for all $k \in \mathbb{Z}$. Then $\{c_kUf_k\}_{k\in\mathbb{Z}}$ is a frame with frame bounds A and B if and only if $\{f_k\}_{k\in\mathbb{Z}}$ is a frame with the same frame bounds A and B.*

Proof. ($\implies$) Assume that $\{c_kUf_k\}_{k\in\mathbb{Z}}$ is a frame, then by (3.1.1) there exists constants $A, B > 0$ such that

$$A\left\|f\right\|^2 \leq \sum_{k\in\mathbb{Z}} |\langle f, c_kUf_k\rangle|^2 \leq B\left\|f\right\|^2, \quad f \in \mathcal{H}. \tag{3.1}$$

Then as U is unitary, its adjoint operator U^* is unitary as well and $\|Uf\| = \|f\| = \|U^*f\|$.

Observe that $|\langle f, c_kUf_k\rangle|^2 = |\overline{c_k}|^2|\langle f, Uf_k\rangle|^2 = |\langle U^*f, f_k\rangle|^2$, thus we can rewrite (3.1) to

$$A\left\|U^*f\right\|^2 \leq \sum_{k\in\mathbb{Z}} |\langle U^*f, f_k\rangle|^2 \leq B\left\|U^*f\right\|^2, \quad f \in \mathcal{H}. \tag{3.2}$$

Since U is unitary, it is a bijection with U^* as inverse. Thus, we can write every $g \in \mathcal{H}$ as $g = U^*f$ for some $f \in \mathcal{H}$, thus making $\{f_k\}_{k\in\mathbb{Z}}$ a frame with the same frame bounds.

The proof is reversible, thus ($\impliedby$) is trivial. □

3.2 Gabor frames

Gabor frames are frames generated by modulating and translating a generator function ϕ often called a window function in signal processing. They appear in many engineering domains, such as image processing [7], and in rheology where they can be used to find the optimal time frequency resolution when studying a mutating material's viscoelastic properties [20]. Furthermore, they can also be used for model selection [1] and in time-frequency analysis of music signals [9]. Common for all these use cases is the time-frequency analysis. This is only natural as Gabor frames are essentially a series of short time Fourier transformations taken on different parts of a function.

In this section, we start by stating the definition of a Gabor frame and a necessary condition for being a Gabor frame. We then show how translation, modulation, dilation, and the Fourier transform affect the frame bounds.

Definition 3.2.1 *Let $\alpha, \beta > 0$. A **Gabor frame** is a frame for $L^2(\mathbb{R})$ of the form $G(\phi, \alpha, \beta) = \{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ where $\phi \in L^2(\mathbb{R})$ is a fixed window function. Furthermore, the frame set of ϕ is given by*

$$\mathfrak{F}(\phi) = \{(\alpha, \beta) \in \mathbb{R}^2_+ : \mathcal{G}(\phi, \alpha, \beta) \text{ is a frame}\}.$$

*Note that $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is called a **Gabor system***

If $\beta = 0$, then $\{T_{n\alpha}\phi\}$ is called a shift invariant system, which can also have frame properties, though it is not a Gabor frame.

We mentioned earlier that frames can be used to reconstruct a function f, like we can with bases. It would of course be ideal if we could just find a basis, but unfortunately, if ϕ is a continuous function with compact support, then $\mathcal{G}(\phi, \alpha, \beta)$ cannot be an orthonormal basis nor a Riesz basis. It can, however, be a frame [4, p. 334]. This is the main motivation for Gabor frame analysis and the Gabor frame set problem.

The Gabor frame set problem consists of finding all parameters (α, β) for a given window function ϕ, such that $\mathcal{G}(\phi, \alpha, \beta)$ is a frame. This problem is only fully solved for a small number of window functions. Though the frame set of an arbitrary window function is unknown, some necessary conditions have been discovered.

Theorem 3.2.2 *[4, p. 266] Let $\phi \in L^2(\mathbb{R})$ and $\alpha, \beta > 0$. If $\alpha\beta > 1$ then $\mathcal{G}(\phi, \alpha, \beta)$ is not a frame for $L^2(\mathbb{R})$.*

This theorem limits which values of α and β we need to consider when trying to characterize the frame set of a given window function. If $\alpha\beta = 1$ and $\mathcal{G}(\phi, \alpha, \beta)$ is a frame, then $\mathcal{G}(\phi, \alpha, \beta)$ is a Riesz basis [4, p. 266], and thus ϕ cannot be continuous and have compact support. We will therefore only consider $\alpha, \beta > 0$ such that $\alpha\beta < 1$.

We now show how translation, modulation, and dilation affect the values of α and β such that $\mathcal{G}(\phi, \alpha, \beta)$ is still a frame.

Theorem 3.2.3 *It holds for $\alpha, \beta, \delta, a, b > 0$ that:*
(i) $(\alpha, \beta) \in \mathfrak{F}(\phi)$ if and only if $(\alpha, \beta) \in \mathfrak{F}(T_a\phi)$ with the same frame bounds
(ii) $(\alpha, \beta) \in \mathfrak{F}(\phi)$ if and only if $(\alpha, \beta) \in \mathfrak{F}(E_b\phi)$ with the same frame bounds
(iii) $(\alpha, \beta) \in \mathfrak{F}(\phi)$ if and only if $(\alpha\delta, \frac{\beta}{\delta}) \in \mathfrak{F}(D_\delta\phi)$ with the same frame bounds.

Proof. ($\Longrightarrow$) We prove all assertions simultaneously. Assume $(\alpha, \beta) \in \mathfrak{F}(\phi)$ then $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is a frame. By Theorem 2.1.5 T_a, E_b, and D_c are unitary operators thus by Lemma 3.1.5 then $\{x_{m,n}T_aE_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$, $\{y_{m,n}E_bE_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$, and $\{z_{m,n}D_cE_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ are frames for $x_{m,n}, y_{m,n}, z_{m,n} \in \mathbb{C}$ with $|x_{m,n}| = |y_{m,n}| = |z_{m,n}| = 1$.

First, observe that by Lemma 2.1.6 it holds that

(i) $T_aE_{m\beta}T_{n\alpha} = e^{-2\pi i m\beta a}E_{m\beta}T_aT_{n\alpha} = e^{-2\pi i m\beta a}E_{m\beta}T_{n\alpha}T_a$

(ii) $E_bE_{m\beta}T_{n\alpha} = E_{m\beta}E_bT_{n\alpha} = E_{m\beta}e^{2\pi i n\alpha b}T_{n\alpha}E_b = e^{2\pi i n\alpha b}E_{m\beta}T_{n\alpha}E_b$

(iii) $D_cE_{m\beta}T_{n\alpha} = E_{m\beta/c}D_cT_{n\alpha} = E_{m\beta/c}T_{n\alpha c}D_c$.

Choose $x_{m,n} = e^{2\pi i m\beta a}$, $y_{m,n} = e^{-2\pi i n\alpha b}$ and $z_{m,n} = 1$. Then

(i) $\{x_{m,n}T_aE_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}} = \{x_{m,n}e^{-2\pi i m\beta a}E_{m\beta}T_{n\alpha}T_a\phi\}_{m,n\in\mathbb{Z}} = \{E_{m\beta}T_{n\alpha}T_a\phi\}_{m,n\in\mathbb{Z}}$

(ii) $\{y_{m,n}E_bE_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}} = \{y_{m,n}e^{2\pi i n\alpha b}E_{m\beta}T_{n\alpha}E_b\phi\}_{m,n\in\mathbb{Z}} = \{E_{m\beta}T_{n\alpha}E_b\phi\}_{m,n\in\mathbb{Z}}$

(iii) $\{z_{m,n}D_cE_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}} = \{z_{m,n}E_{m\beta/c}T_{n\alpha c}D_c\phi\}_{m,n\in\mathbb{Z}} = \{E_{m\beta/c}T_{n\alpha c}D_c\phi\}_{m,n\in\mathbb{Z}}$.

Thus, $\{E_{m\beta}T_{n\alpha}T_a\phi\}_{m,n\in\mathbb{Z}}$, $\{E_{m\beta}T_{n\alpha}E_b\phi\}_{m,n\in\mathbb{Z}}$, and $\{E_{m\beta/c}T_{n\alpha c}D_c\phi\}_{m,n\in\mathbb{Z}}$ are frames and $(\alpha, \beta) \in \mathfrak{F}(T_a\phi)$, $(\alpha, \beta) \in \mathfrak{F}(E_b\phi)$ and $(\alpha c, \frac{\beta}{c}) \in \mathfrak{F}(D_c\phi)$ with the same frame bounds by Lemma 3.1.5.

The proof is revertible, thus ($\Longleftarrow$) is trivial. □

It can be useful to work with the Fourier transform of the window function. The following theorem shows us that this is possible as long as we switch the roles of α and β.

Theorem 3.2.4 *Let $\phi \in L^2(\mathbb{R})$, then*

$$(\alpha, \beta) \in \mathfrak{F}(\phi) \text{ if and only if } (\beta, \alpha) \in \mathfrak{F}(\widehat{\phi}) \text{ with the same frame bounds.}$$

Proof. ($\Longrightarrow$) Assume $(\alpha, \beta) \in \mathfrak{F}(\phi)$ then $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is a frame. By Theorem 2.2.2 we know that the Fourier transform is a unitary operator on $L^2(\mathbb{R})$, thus by Lemma 3.1.5 we know that $\{c_{m,n}\mathcal{F}E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is a frame for $c_{m,n} \in \mathbb{R}$ such that $|c_{m,n}| = 1$. By Theorem 2.2.3 and Lemma 2.1.6 we get

$$c_{m,n}\mathcal{F}E_{m\beta}T_{n\alpha} = c_{m,n}T_{m\beta}\mathcal{F}T_{n\alpha} = c_{m,n}T_{m\beta}E_{-n\alpha}\mathcal{F} = c_{m,n}e^{-2\pi i(-n\alpha)(m\beta)}E_{-n\alpha}T_{m\beta}\mathcal{F}$$

for $n, m \in \mathbb{Z}$. Let $c_{m,n} = e^{-2\pi i n\alpha m\beta}$ then $|c_{m,n}| = 1$ for $n, m \in \mathbb{Z}$, thus

$$\begin{aligned}\{c_{m,n}\mathcal{F}E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}} &= \{e^{-2\pi i n\alpha m\beta}e^{-2\pi i(-n\alpha)(m\beta)}E_{-n\alpha}T_{m\beta}\mathcal{F}\phi\}_{m,n\in\mathbb{Z}} \\ &= \{E_{-n\alpha}T_{m\beta}\mathcal{F}\phi\}_{m,n\in\mathbb{Z}} = \{E_{n\alpha}T_{m\beta}\widehat{\phi}\}_{m,n\in\mathbb{Z}}\end{aligned}$$

and therefore $(\beta, \alpha) \in \mathfrak{F}(\widehat{\phi})$, since $\{c_{m,n}\mathcal{F}E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}} = \{E_{n\alpha}T_{m\beta}\widehat{\phi}\}_{m,n\in\mathbb{Z}}$ is a frame with the same frame bounds by Lemma 3.1.5.

The proof is revertible, thus ($\Longleftarrow$) is trivial. □

3.3 The Zibulski-Zeevi matrix

In this section, we will look at the Zibulski-Zeevi method for computing the frame bounds for a Gabor frame. It is outside the scope of this thesis to prove the Zibulski-Zeevi method, but we will give some intuition as to why the method works.

The Zibulski-Zeevi method only works on **rationally over-sampled Gabor systems** which means that we only look at the Gabor systems $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ where $\alpha\beta = \frac{p}{q} \in \mathbb{Q}$ with $1 \le p \le q$ and $\gcd(p, q) = 1$.

We start by giving some basic definitions of singular values of matrices, which we will use later:

Definition 3.3.1 (Singular values) *[11, p. 328] Let $\boldsymbol{M} \in \mathbb{C}^{m\times n}$ and $q = \min\{m, n\}$. If $\operatorname{rank}\boldsymbol{M} = r \ge 1$, let $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > 0$ be the decreasingly ordered eigenvalues of $(\boldsymbol{M}\boldsymbol{M}^*)^{1/2}$. The **singular values** of $\boldsymbol{M}$ are then*

$$\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > 0 \quad \textit{and} \quad \sigma_{r+1} = \cdots = \sigma_q = 0.$$

Note that $\sigma_1^2 \ge \sigma_2^2 \ge \cdots \ge \sigma_r^2$ are the positive eigenvalues of $\boldsymbol{M}^*\boldsymbol{M}$ and $\boldsymbol{M}\boldsymbol{M}^*$.

Theorem 3.3.2 (Singular value decomposition) *[11, p. 329] Let $\boldsymbol{M} \in \mathbb{C}^{m\times n}$ be non-zero and let $r = \operatorname{rank}\boldsymbol{M}$. Let $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > 0$ be the positive singular values of $\boldsymbol{M}$ and define*

$$\Sigma_r = \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_r \end{bmatrix} \in \mathbb{C}^{r\times r}.$$

Then there exist unitary matrices $\boldsymbol{U} \in \mathbb{C}^{m\times m}$ and $\boldsymbol{V} \in \mathbb{C}^{n\times n}$ such that

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*,$$

called the singular value decomposition of $\boldsymbol{M}$, where

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_r & \boldsymbol{0}_{r\times(n-r)} \\ \boldsymbol{0}_{(m-r)\times r} & \boldsymbol{0}_{(m-r)\times(n-r)} \end{bmatrix} \in \mathbb{C}^{m\times n}$$

is the same size as $\boldsymbol{M}$.

Before introducing the Zibulski-Zeevi matrix, we want to show how the frame bounds of a finite frame $\{\boldsymbol{v}_k\}_{k=1}^n$ relates to the singular values of the matrix $\boldsymbol{M} = \begin{bmatrix}\boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n\end{bmatrix}$.

Theorem 3.3.3 *Let $\{\boldsymbol{v}_k\}_{k=1}^n$ be a finite frame for a m-dimensional vector space $W \subseteq \mathbb{C}^m$ and consider an $m\times n$ matrix $\boldsymbol{M} = \begin{bmatrix}\boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n\end{bmatrix}$. If $\operatorname{rank}\boldsymbol{M} = r \ge 1$, then $\boldsymbol{M}$ has singular values $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_q = 0$, where $q = \min\{m, n\}$. Then the frame bounds A and B for the frame $\{\boldsymbol{v}_k\}_{k=1}^n$ are given by $A = \sigma_r^2$ and $B = \sigma_1^2$.*

Proof. Let $\boldsymbol{v} \in W$ and let $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$ be the singular value decomposition of $\boldsymbol{M}$. Observe that

$$\sum_{k=1}^{n} |\langle \boldsymbol{v}, \boldsymbol{v}_k\rangle|^2 = \|\boldsymbol{M}^*\boldsymbol{v}\|^2 = \|(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*)^*\boldsymbol{v}\|^2 = \|\boldsymbol{V}\boldsymbol{\Sigma}^*\boldsymbol{U}^*\boldsymbol{v}\|^2 =: (*).$$

Let $\boldsymbol{w} = \boldsymbol{U}^*\boldsymbol{v}$. Since V is unitary it holds that

$$(*) = \|\boldsymbol{\Sigma}^*\boldsymbol{w}\|^2 = \sum_{i=1}^{q} |\sigma_i w_i|^2 = \sum_{i=1}^{q} \sigma_i^2 |w_i|^2.$$

We now see that

$$\sum_{i=1}^{q} \sigma_i^2 |w_i|^2 \leq \sigma_1^2 \sum_{i=1}^{q} |w_i|^2 = \sigma_1^2 \|\boldsymbol{w}\|^2 = \sigma_1^2 \|\boldsymbol{U}^*\boldsymbol{v}\|^2 = \sigma_1^2 \|\boldsymbol{v}\|^2$$

and likewise

$$\sum_{i=1}^{q} \sigma_i^2 |w_i|^2 \geq \sigma_r^2 \sum_{i=1}^{q} |w_i|^2 = \sigma_r^2 \|\boldsymbol{w}\|^2 = \sigma_r^2 \|\boldsymbol{U}^*\boldsymbol{v}\|^2 = \sigma_r^2 \|\boldsymbol{v}\|^2 .$$

Thus, we get

$$\sigma_r^2 \|\boldsymbol{v}\|^2 \leq \sum_{k=1}^{n} |\langle \boldsymbol{v}, \boldsymbol{v}_k \rangle|^2 \leq \sigma_1^2 \|\boldsymbol{v}\|^2$$

which by Definition 3.1.1 shows that $A = \sigma_r^2$ and $B = \sigma_1^2$ are frame bounds for the frame $\{\boldsymbol{v}_k\}_{k=1}^{n}$. □

This shows that we can use singular values to find the frame bounds for finite frames. It is therefore not unreasonable to think that a similar result can be found in the case of infinite frames like Gabor frames. The Zibulski-Zeevi method is based on this idea but uses a specific matrix called the Zibulski-Zeevi matrix. We thus start by introducing the Zak-transformation which is used to construct the Zibulski-Zeevi matrix.

Definition 3.3.4 *[4, p. 330] Let $\lambda > 0$ be a fixed parameter and $f \in L^2(\mathbb{R})$. We define the **Zak transform** of f as*

$$(Z_\lambda f)(t, \nu) = \lambda^{1/2} \sum_{k \in \mathbb{Z}} f(\lambda(t-k)) e^{2\pi i k \nu} \qquad t, \nu \in \mathbb{R}.$$

We can now define the Zibulski-Zeevi matrix

Definition 3.3.5 *[4, p. 334] The **Zibulski-Zeevi matrix** associated with the Gabor system $\{E_{m\beta} T_{n\alpha} \phi\}_{m,n \in \mathbb{Z}}$ with $\alpha\beta = \frac{p}{q}$ is defined as:*

$$\Phi^\phi(t, \nu) = p^{-1/2} \left[(Z_{\frac{1}{\beta}} \phi) \left(t - \ell \frac{p}{q}, \nu + \frac{k}{p} \right) \right]_{k \in \mathbb{Z}_p;\ \ell \in \mathbb{Z}_q} \qquad \text{for a.e } t, \nu \in \mathbb{R}.$$

The Zibulski-Zeevi matrix is a $p \times q$ matrix, where $\alpha\beta = \frac{p}{q} \in \mathbb{Q}$. Furthermore, it is 1-periodic in both t and ν i.e. $\Phi^\phi(t+i, \nu+j) = \Phi^\phi(t, \nu)$ for any $i, j \in \mathbb{Z}$.

We will now see how we can use the Zibulski-Zeevi matrix to compute the frame bounds of a rationally over-sampled Gabor frame. For a proof of Theorem 3.3.6, we refer to [16] and [21].

Theorem 3.3.6 *[4, p. 335] Assume $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is a rationally over-sampled Gabor system (i.e. $\alpha\beta = \frac{p}{q} \in \mathbb{Q}$ with $1 \le p \le q$ and $\gcd(p,q) = 1$) and let $\Phi^\phi(t,\nu)$ be the Zibulski-Zeevi matrix associated with $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$. Furthermore, let $\sigma_1(\Phi^\phi(t,\nu)) \ge \sigma_2(\Phi^\phi(t,\nu)) \ge \cdots \ge \sigma_r(\Phi^\phi(t,\nu))$ with $r = \operatorname{rank}\Phi^\phi(t,\nu)$ be the singular values of $\Phi^\phi(t,\nu)$ for all $t,\nu \in [0,1[$.*

Then $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is a frame with frame bounds

$$A = \operatorname*{ess\,inf}_{t,\nu\in[0,1[} \sigma_r(\Phi^\phi(t,\nu))^2 \qquad \text{and} \qquad B = \operatorname*{ess\,sup}_{t,\nu\in[0,1[} \sigma_1(\Phi^\phi(t,\nu))^2$$

if and only if

$$0 < \operatorname*{ess\,inf}_{t,\nu\in[0,1[} \sigma_r(\Phi^\phi(t,\nu))^2 \le \operatorname*{ess\,sup}_{t,\nu\in[0,1[} \sigma_1(\Phi^\phi(t,\nu))^2 < \infty.$$

We see that this theorem is similar to Theorem 3.3.3, but for the infinite frame case. The downside is that we introduce two indeterminates t and ν. Since the size of the Zibulski-Zeevi matrix is $p \times q$, it can be very computationally demanding to calculate its singular values for large p and q with $\alpha\beta = \frac{p}{q} \in \mathbb{Q}$.

If we are not interested in the frame bounds and our window function $\phi \in \mathcal{S}_0$, where $\mathcal{S}_0$ is the Feichtinger's algebra [3], then the following simplification was obtained by Lyubarskii and Nes in [19]:

Corollary 3.3.7 *[4, p. 335] Let $\phi \in \mathcal{S}_0$ and assume $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is a rationally over-sampled Gabor system with $\alpha\beta = \frac{p}{q} \in \mathbb{Q}$. Then the Gabor system $\{E_{m\beta}T_{n\alpha}\phi\}_{m,n\in\mathbb{Z}}$ is a frame if and only if the $p \times q$ matrix*

$$\left[\sum_{n\in\mathbb{Z}} \phi(t + \alpha\ell - \alpha q n + k/\beta) e^{2\pi i n \alpha q \nu}\right]_{k\in\mathbb{Z}_p;\ \ell\in\mathbb{Z}_q}$$

has rank p for all $(t,\nu) \in [0,\alpha/p[\times[0,1/\alpha[$.

Observe that to disprove the frame property, we would only need to find a single point $(t,\nu) \in [0,\alpha/p[\times[0,1/\alpha[$, where the matrix does not have rank p. This property was used by Lemvig and Nielsen to disprove the frame set conjecture for B-splines in [17]. For a proof of Corollary 3.3.7 we refer to [19, p. 492]. Lemvig and Nielsen also showed the following family of obstructions to the frame set property of $\mathcal{G}(Q_m,\alpha,\beta)$ based on the Zak transform in [17, p. 1447].

Theorem 3.3.8 *Let $m \in \mathbb{N}$, $\alpha > 0$, and $\beta > \frac{3}{2}$. Let p and q be positive and relatively prime and define $F(x) = x - \lfloor x + \frac{1}{2} \rfloor \in (-\frac{1}{2}, \frac{1}{2}]$. If*

$$\alpha\beta = \frac{p}{q}, \quad \text{and} \quad |F(b)| \le \frac{1}{\beta q}, \tag{3.3}$$

then $\mathcal{G}(Q_m,\alpha,\beta)$ is not a frame.

Note that it is not known if this gives the optimal range of the non-frame property, though Lemvig and Nielsen believe that it is optimal when $p = 1$.

Unlike the Zibulski-Zeevi and the Lyubarskii-Nes methods, this method is easy to compute and is still able to guarantee that $\mathcal{G}(Q_m,\alpha,\beta)$ is not a frame for some α and β.

3.4 Shift invariant spaces

The frame property of Gabor frames is tied together with shift invariant spaces, a fact that the Ghosh-Selvan method uses. The main result of this section shows exactly how the two are connected. We start by introducing an important theorem that will be used for the main result in this section.

Theorem 3.4.1 *[4, p. 265] The Gabor system $\{E_{m\beta}T_{n\alpha}\phi\}$ is a frame with bounds $A, B > 0$ if and only if for a.e $x \in \mathbb{R}$ the inequalities*

$$\beta A \sum_{k\in\mathbb{Z}} |c_k|^2 \leq \sum_{n\in\mathbb{Z}} \left| \sum_{k\in\mathbb{Z}} \overline{\phi(x - n\alpha - k/\beta)} c_k \right|^2 \leq \beta B \sum_{k\in\mathbb{Z}} |c_k|^2 \tag{3.4}$$

hold for all $\{c_k\}_{k\in\mathbb{Z}} \in \ell^2(\mathbb{Z})$.

To present the Ghosh-Selvan method in the next chaptor we first need to introduce the shift-invariant space and an important theorem linking shift-invariant spaces and Gabor frames. We start by defining the shift-invariant space.

Definition 3.4.2 *[13, p. 1] The **shift-invariant** space of a generator ϕ where $h > 0$ is given by*

$$V_h(\phi) = \left\{ f \in L^2(\mathbb{R}) \mid f = \sum_{k\in\mathbb{Z}} c_k T_{hk}\phi \text{ for some } \{c_k\}_{k\in\mathbb{Z}} \in \ell^2(\mathbb{Z}) \right\}.$$

*The generator ϕ is said to be a **stable generator** for $V_h(\phi)$ if $\{T_{hk}\phi\}_{k\in\mathbb{Z}}$ is a (Riezs) basis for $V_h(\phi)$ and there exist constants $A, B > 0$ such that*

$$A \sum_{k\in Z} |c_k|^2 \leq \left\| \sum_{k\in Z} c_k T_{hk}\phi \right\|^2 \leq B \sum_{k\in Z} |c_k|^2.$$

*A set $\Delta = \{x_n : n \in \mathbb{Z}\}$ of real numbers is said to be a **set of stable sampling** for $V_h(\phi)$ if there exist constants $C, D > 0$ called **sampling bounds** such that*

$$C \|f\|^2 \leq \sum_{n\in\mathbb{Z}} |f(x_n)|^2 \leq D \|f\|^2, \quad f \in V_h(\phi). \tag{3.5}$$

Intuitively a shift invariant space V_h is a subspace of $L^2(\mathbb{R})$ such that for any $f \in V_h(\phi)$ it holds that $f(\,\cdot\, + h) \in V_h(\phi)$.

We need the following important lemma from [13, p. 1].

Lemma 3.4.3 *[13, p. 1] A function ϕ is a stable generator for $V_h(\phi)$ if and only if*

$$0 < \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w + \beta j)|^2 \right) \leq \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w + \beta j)|^2 \right) < \infty.$$

This gives a sufficient and necessary condition to determine if a function is a stable generator.

We need Parseval's equation for the next proof, which we will give as a lemma

Lemma 3.4.4 (Parseval's equation) *[6, p. 127] Let $f \in L^2(\mathbb{R})$ with Fourier coefficients c_k for $k \in \mathbb{Z}$ then*

$$\frac{1}{2\pi} \int_{\pi}^{\pi} |f(x)|^2 \, dx = \sum_{k \in \mathbb{Z}} |c_k|^2.$$

Now we can prove the following lemma.

Lemma 3.4.5 *Let $F(w) = \sum_{k \in \mathbb{Z}} c_k e^{-2\pi i \frac{k}{\beta} w}$ then F is β-periodic and*

$$\int_0^\beta |F(w)|^2 \, dw = \beta \|\{c_k\}_{k \in \mathbb{Z}}\|^2.$$

Proof. Observe that

$$F(w+\beta) = \sum_{k \in \mathbb{Z}} c_k e^{-2\pi i \frac{k}{\beta}(w+\beta)} = \sum_{k \in \mathbb{Z}} c_k e^{-2\pi i \frac{k}{\beta} w} e^{-2\pi i \frac{k}{\beta}\beta} = \sum_{k \in \mathbb{Z}} c_k e^{-2\pi i \frac{k}{\beta} w} = F(w)$$

which shows that F is β-periodic.

Let $f := \sum_{k \in \mathbb{Z}} c_k e^{ikx}$ and note that $F(w) = f\left(\frac{-2\pi}{\beta} w\right)$. Then by integration by substitution with $t = \frac{-2\pi}{\beta} w$ we get:

$$\int_{-\pi}^{\pi} |f(t)|^2 \, dt = \int_{\beta/2}^{-\beta/2} \left| f\left(\frac{-2\pi}{\beta} w\right)\right|^2 \left(\frac{-2\pi}{\beta}\right) \, dw = \frac{2\pi}{\beta} \int_{-\beta/2}^{\beta/2} |F(w)|^2 \, dw. \tag{3.6}$$

Then by using Lemma 3.4.4 (Parseval's equation), we see that

$$\begin{aligned}\|\{c_k\}_{k \in \mathbb{Z}}\|^2 &= \sum_{k \in \mathbb{Z}} |c_k|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |f(t)|^2 \, dt \overset{(3.6)}{=} \frac{1}{2\pi} \frac{2\pi}{\beta} \int_{-\beta/2}^{\beta/2} |F(w)|^2 \, dw \\ &= \frac{1}{\beta} \int_0^\beta |F(w)|^2 \, dw\end{aligned}$$

since F is β-periodic. Consequently, we get:

$$\int_0^\beta |F(w)|^2 \, dw = \beta \|\{c_k\}_{k \in \mathbb{Z}}\|^2.$$

□

Lemma 3.4.6 *Let ϕ be a stable generator for $V_{1/\beta}(\phi)$. Then for every $f = \sum c_k T_{\frac{k}{\beta}} \phi \in V_{1/\beta}(\phi)$ it holds that:*

$$\beta \operatorname*{ess\,inf}_{w \in [0,\beta]} \left(\sum_{k \in \mathbb{Z}} \left| \widehat{\phi}(w + k\beta) \right|^2 \right) \|\{c_k\}_{k \in \mathbb{Z}}\|^2 \leq \|f\|^2$$

$$\|f\|^2 \leq \beta \operatorname*{ess\,sup}_{w \in [0,\beta]} \left(\sum_{k \in \mathbb{Z}} \left| \widehat{\phi}(w + k\beta) \right|^2 \right) \|\{c_k\}_{k \in \mathbb{Z}}\|^2.$$

Proof. Let $F(w) := \sum_{k\in\mathbb{Z}} c_k e^{-2\pi i \frac{k}{\beta} w}$. By Theorem 2.2.3, it holds that:

$$\begin{aligned}
\|f\|^2 = \|\widehat{f}\|^2 &= \|\mathcal{F}\sum_{k\in\mathbb{Z}} c_k T_{\frac{k}{\beta}}\phi\|^2 = \|\sum_{k\in\mathbb{Z}} c_k E_{-\frac{k}{\beta}}\widehat{\phi}\|^2 \\
&= \int_{-\infty}^{\infty} \left|\sum_{k\in\mathbb{Z}} c_k e^{-2\pi i\frac{k}{\beta}w}\widehat{\phi}(w)\right|^2 dw \\
&= \sum_{j\in\mathbb{Z}} \int_{j\beta}^{(j+1)\beta} \left|\sum_{k\in\mathbb{Z}} c_k e^{-2\pi i\frac{k}{\beta}w}\right|^2 |\widehat{\phi}(w)|^2 \, dw \\
&= \sum_{j\in\mathbb{Z}} \int_{j\beta}^{(j+1)\beta} |F(w)|^2 |\widehat{\phi}(w)|^2 \, dw \\
&= \sum_{j\in\mathbb{Z}} \int_{0}^{\beta} |F(w+\beta j)|^2 |\widehat{\phi}(w+\beta j)|^2 \, dw.
\end{aligned}$$

Then by Lemma 3.4.5 we see that:

$$\begin{aligned}
\|f\|^2 &= \sum_{j\in\mathbb{Z}} \int_{0}^{\beta} |F(w)|^2 |\widehat{\phi}(w+\beta j)|^2 \, dw \\
&= \int_{0}^{\beta} |F(w)|^2 \sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2 \, dw. \qquad (3.7)
\end{aligned}$$

Lastly, using Lemma 3.4.5 once more we get that:

$$\begin{aligned}
\operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \|\{c_k\}_{k\in\mathbb{Z}}\|^2 &= \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \int_0^\beta |F(w)|^2 \, dw \\
&\leq \int_0^\beta |F(w)|^2 \sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2 \, dw \\
&\overset{(3.7)}{=} \|f\|^2 \\
&\leq \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \int_0^\beta |F(w)|^2 \, dw \\
&= \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \|\{c_k\}_{k\in\mathbb{Z}}\|^2 . \quad \square
\end{aligned}$$

Lemma 3.4.6 gives rise to the following theorem about how the sampling bounds and frame bounds are connected.

Theorem 3.4.7 *Let ϕ be a stable generator for $V_{1/\beta}(\phi)$. If $x+\alpha\mathbb{Z} = \{x+n\alpha \mid n\in\mathbb{Z}\}$ is a stable sampling set for $V_{1/\beta}(\phi)$ with sampling bounds C and D for all $x\in\mathbb{R}$. Then $\{E_{m\beta}T_{n\alpha}\phi\}$ is a frame with positive frame bounds*

$$A = C \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right), \qquad B = D \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right).$$

Proof. Assuming that $x + \alpha\mathbb{Z}$ is a stable sampling set for $V_{1/\beta}(\phi)$ then

$$C\|f\|^2 \leq \sum_{n\in\mathbb{Z}} |f(x+n\alpha)|^2 \leq D\|f\|^2.$$

Let $f = \sum_{k\in\mathbb{Z}} c_k T_{k/\beta}\phi \in V_{1/\beta}(\phi)$. As ϕ is a stable generator for $V_{1/\beta}(\phi)$ it holds by Lemma 3.4.6 that:

$$\begin{aligned} C \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2 \leq C\|f\|^2 \leq \sum_{n\in\mathbb{Z}} |f(x+n\alpha)|^2 \leq D\|f\|^2 \\ \leq D \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2. \end{aligned}$$

Consequently, as

$$\sum_{n\in\mathbb{Z}} |f(x+n\alpha)|^2 = \sum_{n\in\mathbb{Z}} \left|\sum_{k\in\mathbb{Z}} c_k T_{k/\beta}\phi(x-n\alpha)\right|^2 = \sum_{n\in\mathbb{Z}} \left|\sum_{k\in\mathbb{Z}} \overline{\phi(x-n\alpha-k/\beta)} c_k\right|^2,$$

we see that

$$\begin{aligned} C \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2 \leq \sum_{n\in\mathbb{Z}} |\sum_{k\in\mathbb{Z}} \overline{\phi(x-n\alpha-k/\beta)} c_k|^2 \\ \leq D \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2. \end{aligned}$$

Hence, by Theorem 3.4.1 $\{E_{m\beta}T_{n\alpha}\phi\}$ is a frame and there exist frame bounds A and B such that

$$C \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2 = \beta A \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2,$$

and

$$D \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \beta \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2 = \beta B \left\|\{c_k\}_{k\in\mathbb{Z}}\right\|^2.$$

As ϕ is a stable generator we know from Lemma 3.4.3 that

$$0 < \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \leq \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) < \infty.$$

This combined with the fact that C and D are positive, shows that A and B are positive and finite. Consequently, we see that

$$A = C \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right) \text{ and } B = D \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{j\in\mathbb{Z}} |\widehat{\phi}(w+\beta j)|^2\right).$$

□

This shows that if we can find the sampling bounds for $V_{1/\beta}(\phi)$ then we can also find frame bounds for $\mathcal{G}(\phi, \alpha, \beta)$, though it should be noted that these frame bounds are not necessarily optimal.

Chapter 4

Ghosh-Selvan method

The following section relies heavily on section 2 in [13, pp. 4–7], where Riya Ghosh and A. Antony Selvan derived a new method for computing the frame set and frame bounds for a window function ϕ in a specific subspace of $L^2(\mathbb{R})$. The method works by finding sampling bounds for $V_{1/\beta}(\phi)$ and using Theorem 3.4.7 to convert them to frame bounds. As Theorem 3.4.7 also gives conditions for when $\mathcal{G}(\phi, \alpha, \beta)$ is a frame, we can also use it to compute the frame set of ϕ.

We have implemented this method numerically and by plotting the frame bounds, we found that the frame bounds were sometimes switched.

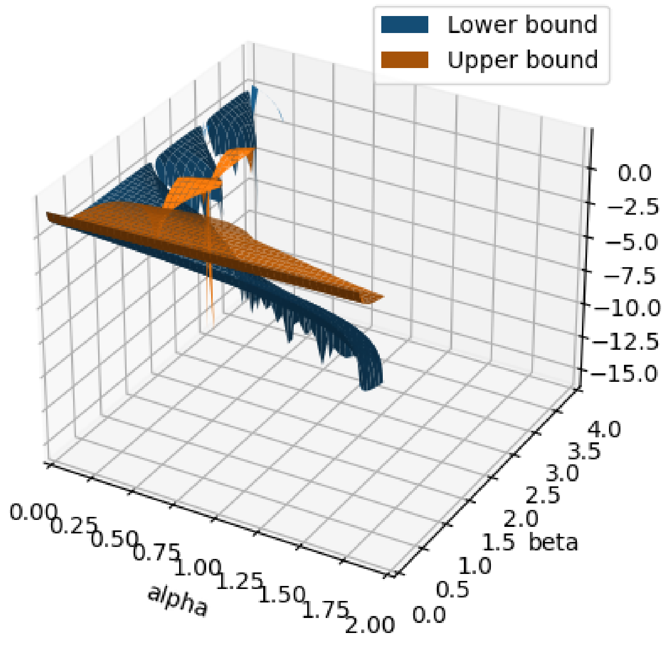

Figure 4.1: Plot of the frame bounds for Q_2 computed with the method proposed in [13]. The z-axis is $\log_{10}$-scaled. Note that the upper frame bounds sometimes fall below the lower frame bounds.

As we could not find any mistakes in our code, we concluded that there was a mistake in [13], and we discovered that Theorem 1.1 in [13, p. 2] was used incorrectly. In this chapter, we will show how to derive the Ghosh-Selvan method and use

Theorem 3.4.7 to propose a correct way to compute a lower and upper frame bound.

4.1 Derivation of the method

We start by defining the class of window functions that we will be working with.

Definition 4.1.1 *Let $\mathcal{A}$ denote the class of real-valued continuous functions ϕ satisfying the following conditions:*
(i) ϕ is differentiable except at a finite number of points
(ii) For some $\epsilon > 0$, $\phi^{(s)}(x) = \mathcal{O}(|x|^{-0.5-\epsilon})$ as $x \to \pm\infty$, $s = 0, 1$
(iii) For all $h > 0, \operatorname{ess\,sup}_{w\in[0,1/h]} \sum_{n\in\mathbb{Z}}(w + \frac{n}{h})^2|\widehat{\phi}(w + \frac{n}{h})|^2 < \infty$.

As any $\phi \in \mathcal{A}$ is continuous, it is measurable and bounded on any bounded interval. By (ii) there exist some $a, c \in \mathbb{R}_{>0}$ such that for $|x| \geq a$ then $|\phi(x)| \leq c|x|^{-0.5-\epsilon}$, hence:

$$\begin{aligned}\int |\phi|^2 &\leq \int_{-a}^{a} |\phi|^2 + \int_{-\infty}^{-a} |c \cdot |x|^{-0.5-\epsilon}|^2 \, dx + \int_{a}^{\infty} |c \cdot |x|^{-0.5-\epsilon}|^2 \, dx \\ &= \int_{-a}^{a} |\phi|^2 + c^2 \int_{-\infty}^{-a} |x|^{-1-2\epsilon} \, dx + c^2 \int_{a}^{\infty} |x|^{-1-2\epsilon} \, dx \\ &\leq 2a \max_{w\in[-a,a]} |\phi(w)|^2 + c^2 \int_{\mathbb{R}\setminus[-a,a]} |x|^{-1-2\epsilon} \, dx < \infty.\end{aligned}$$

Thus, $\mathcal{A} \subset L^2(\mathbb{R})$, and hence the Fourier transform is well defined for $\phi \in \mathcal{A}$ and thus (iii) makes sense. We can therefore consider

$$B_{\phi,h}(w) = \frac{\sum_{n\in\mathbb{Z}}(w + \frac{n}{h})^2|\widehat{\phi}(w + \frac{n}{h})|^2}{\sum_{n\in\mathbb{Z}} |\widehat{\phi}(w + \frac{n}{h})|^2} \quad \text{and} \quad M_{\phi,h} = \operatorname*{ess\,sup}_{\omega\in[0,1/h]} B_{\phi,h}(w). \tag{4.1}$$

The idea of this method is to find sampling bounds for a stable generator $\phi \in \mathcal{A}$ for $V_h(\phi)$, and then convert them to frame bounds using Theorem 3.4.7. In the rest of this section, we follow [13] with a detailed explanation of the proofs.

Theorem 4.1.2 *If $\phi \in \mathcal{A}$ is a stable generator for $V_h(\phi)$, then*

$$\|f'\| \leq 2\pi\sqrt{M_{\phi,h}}\, \|f\| ,$$

for every $f \in V_h(\phi)$.

Proof. Since $f \in V_h(\phi)$, we know that we can write $f = \sum_{k\in\mathbb{Z}} d_k T_{hk}\phi$. Furthermore, we know from the proof of Lemma 2.2 in [18, p. 623] that $f' = \sum_{k\in\mathbb{Z}} d_k T_{hk}\phi'$. Thus, by Theorem 2.2.3 and Theorem 2.2.4 it holds that

$$\begin{aligned}
\|f'\|^2 = \|\widehat{f'}\|^2 &= \left\|\mathcal{F}\sum_{k\in\mathbb{Z}} d_k T_{hk}\phi'\right\|^2 = \left\|\sum_{k\in\mathbb{Z}} d_k E_{-hk}\mathcal{F}\phi'\right\|^2 \\
&= \int_{-\infty}^{\infty} \left|\sum_{k\in\mathbb{Z}} d_k e^{-2\pi i hk\omega}\widehat{\phi'}(\omega)\right|^2 d\omega \\
&= \int_{-\infty}^{\infty} \left|\sum_{k\in\mathbb{Z}} d_k e^{-2\pi i hk\omega}(2\pi i\omega)\widehat{\phi}(\omega)\right|^2 d\omega \\
&= \sum_{\ell\in\mathbb{Z}} \int_{\frac{\ell}{h}}^{\frac{\ell+1}{h}} \left|\sum_{k\in\mathbb{Z}} d_k e^{-2\pi i hk\omega}(2\pi i\omega)\widehat{\phi}(\omega)\right|^2 d\omega \\
&= 4\pi^2 \sum_{\ell\in\mathbb{Z}} \int_0^{\frac{1}{h}} \left|\sum_{k\in\mathbb{Z}} d_k e^{-2\pi i hk(\omega+\ell/h)}\right|^2 \left|\left(\omega+\frac{\ell}{h}\right)\widehat{\phi}\left(\omega+\frac{\ell}{h}\right)\right|^2 d\omega.
\end{aligned}$$

We define $m_f(\omega) := \sum_{k\in\mathbb{Z}} d_k e^{-2\pi i hk\omega}$, which is $\frac{1}{h}$-periodic since

$$m_f\left(\omega+\frac{1}{h}\right) = \sum_{k\in\mathbb{Z}} d_k e^{-2\pi i hk(\omega+1/h)} = e^{-2\pi i hk\omega}e^{-2\pi i k} = e^{-2\pi i hk\omega} = m_f(\omega).$$

We recognize that $\sum_{\ell\in\mathbb{Z}} \left|(\omega+\frac{\ell}{h})\widehat{\phi}(\omega+\frac{\ell}{h})\right|^2 = B_{\phi,h}(\omega)\sum_{\ell\in\mathbb{Z}}\left|\widehat{\phi}(\omega+\frac{\ell}{h})\right|^2$, thus as $M_{\phi,h} = \operatorname{ess\,sup}_{\omega\in[0,1/h]} B_{\phi,h}(\omega)$ and m_f is $\frac{1}{h}$ periodic it holds that:

$$\begin{aligned}
\|f'\|^2 &= 4\pi^2 \int_0^{\frac{1}{h}} |m_f(\omega)|^2 B_{\phi,h}(\omega) \sum_{\ell\in\mathbb{Z}} \left|\widehat{\phi}(\omega+\frac{\ell}{h})\right|^2 d\omega \\
&\leq 4\pi^2 M_{\phi,h} \int_0^{\frac{1}{h}} |m_f(\omega)|^2 \sum_{\ell\in\mathbb{Z}} \left|\widehat{\phi}(\omega+\frac{\ell}{h})\right|^2 d\omega \\
&= 4\pi^2 M_{\phi,h} \sum_{\ell\in\mathbb{Z}} \int_0^{\frac{1}{h}} |m_f(\omega)|^2 \left|\widehat{\phi}(\omega+\frac{\ell}{h})\right|^2 d\omega \\
&= 4\pi^2 M_{\phi,h} \sum_{\ell\in\mathbb{Z}} \int_{\frac{\ell}{h}}^{\frac{\ell+1}{h}} |m_f(\omega)|^2 \left|\widehat{\phi}(\omega)\right|^2 d\omega \\
&= 4\pi^2 M_{\phi,h} \int_{-\infty}^{\infty} |m_f(\omega)|^2 \left|\widehat{\phi}(\omega)\right|^2 d\omega \\
&= 4\pi^2 M_{\phi,h} \int_{-\infty}^{\infty} \left|\sum_{k\in\mathbb{Z}} d_k e^{-2\pi i hk\omega}\widehat{\phi}(\omega)\right|^2 d\omega \\
&= 4\pi^2 M_{\phi,h}\|\widehat{f}\|^2 = 4\pi^2 M_{\phi,h}\|f\|^2.
\end{aligned}$$

This shows that $\|f'\| \leq 2\pi\sqrt{M_{\phi,h}}\,\|f\|$. □

We need the approximation operator operator for the next proofs, which we define as the following.

Definition 4.1.3 *Let P be an orthogonal projection of $L^2(\mathbb{R})$ onto $V_h(\phi)$. Then we define the* ***approximation operator*** *A on $V_h(\phi)$ by:*

$$Af = P\left(\sum_{n\in\mathbb{Z}} f(x_n)\chi_{[y_n,y_{n+1}]}\right)$$

with $y_n = \frac{x_{n-1}+x_n}{2}$, where $\{x_n\}$ is a set of stable sampling.[1]

We need the following two lemmas for the next proof:

Lemma 4.1.4 (Wirtinger's inequality) *[15, p. 183] If $g, g' \in L^2[a,b]$ such that $g(a) = 0$ or $g(b) = 0$. Then*

$$\int_a^b |g(x)|^2\, dx \leq \frac{4}{\pi^2}(b-a)^2 \int_a^b |g'(x)|^2\, dx.$$

Lemma 4.1.5 (Neumann's theorem) *[4, p. 51] If $U : X \to X$ is bounded and $\|I-U\| < 1$, then U is invertible, and*

$$U^{-1} = \sum_{k=0}^{\infty}(I-U)^k.$$

Furthermore,

$$\left\|U^{-1}\right\| \leq \frac{1}{1-\|I-U\|}.$$

Now we have the tools to prove our next theorems.

Theorem 4.1.6 *Let $\phi \in \mathcal{A}$ be a stable generator for $V_h(\phi)$. If $\delta = \sup_{n\in\mathbb{Z}}(x_{n+1} - x_n) < \frac{1}{2\sqrt{M_{\phi,h}}}$, then for every $f \in V_h(\phi)$*

$$\|f - Af\| \leq 2\delta\sqrt{M_{\phi,h}}\,\|f\|. \tag{4.2}$$

Consequently, A is a bounded invertible operator on $V_h(\phi)$ with bounds

$$\|Af\| \leq (1 + 2\delta\sqrt{M_{\phi,h}})\,\|f\| \tag{4.3}$$

and

$$\left\|A^{-1}f\right\| \leq (1 - 2\delta\sqrt{M_{\phi,h}})^{-1}\,\|f\|. \tag{4.4}$$

Proof. Let P be an orthogonal projection of $L^2(\mathbb{R})$ onto $V_h(\phi)$ and let $f \in V_h(\phi)$. By Lemma 2.1.3 we know that $Pf = f$ and P is linearly bounded with $\|P\| = 1$. Since the intervals $[y_n, y_{n+1}]$ are non-overlapping and $\bigcup_{n\in\mathbb{Z}}[y_n, y_{n+1}] = \mathbb{R}$, we can

[1]Note that the indices in this definition differs from the one in [13].

write $f = \sum_{n\in\mathbb{Z}} f\chi_{[y_n,y_{n+1}]}$. Observe that

$$\begin{aligned}
\|f - Af\|^2 &= \left\| Pf - P\left(\sum_{k\in\mathbb{Z}} f(x_k)\chi_{[y_k,y_{k+1}]}\right)\right\|^2 \\
&\leq \|P\|^2 \left\| f - \sum_{k\in\mathbb{Z}} f(x_k)\chi_{[y_k,y_{k+1}]}\right\|^2 \\
&= \left\| \sum_{k\in\mathbb{Z}} f\chi_{[y_k,y_{k+1}]} - \sum_{k\in\mathbb{Z}} f(x_k)\chi_{[y_k,y_{k+1}]}\right\|^2 \\
&= \left\| \sum_{k\in\mathbb{Z}} (f - f(x_k))\chi_{[y_k,y_{k+1}]}\right\|^2 \\
&= \int_{-\infty}^{\infty} \left| \sum_{k\in\mathbb{Z}} (f(x) - f(x_k))\chi_{[y_k,y_{k+1}]}\right|^2 \, dx \\
&= \sum_{n\in\mathbb{Z}} \int_{y_n}^{y_{n+1}} \left| \sum_{k\in\mathbb{Z}} (f(x) - f(x_k))\chi_{[y_k,y_{k+1}]}\right|^2 \, dx \\
&= \sum_{n\in\mathbb{Z}} \int_{y_n}^{y_{n+1}} |f(x) - f(x_n)|^2 \, dx \\
&= \sum_{n\in\mathbb{Z}} \left(\int_{y_n}^{x_n} |f(x) - f(x_n)|^2 \, dx + \int_{x_n}^{y_{n+1}} |(f(x) - f(x_n))|^2 \, dx \right).
\end{aligned}$$

since $y_n = \frac{x_{n-1}+x_n}{2}$, and thus $y_n < x_n < y_{n+1}$. Let $g = f - f(x_n)$ then $g(x_n) = 0$ and $g' = f'$. Thus, we can use Lemma 4.1.4 (Wirtinger's inequality) to get

$$\begin{aligned}
\|f - Af\|^2 &= \sum_{n\in\mathbb{Z}} \left(\int_{y_n}^{x_n} |f(x) - f(x_n)|^2 \, dx + \int_{x_n}^{y_{n+1}} |(f(x) - f(x_n))|^2 \, dx \right) \\
&= \sum_{n\in\mathbb{Z}} \left(\int_{y_n}^{x_n} |g(x)|^2 \, dx + \int_{x_n}^{y_{n+1}} |g(x)|^2 \, dx \right) \\
&\leq \sum_{n\in\mathbb{Z}} \left(\frac{4}{\pi^2}(x_n - y_n)^2 \int_{y_n}^{x_n} |g'(x)|^2 \, dx + \frac{4}{\pi^2}(y_{n+1} - x_n)^2 \int_{x_n}^{y_{n+1}} |g'(x)|^2 \, dx \right) \\
&\leq \sum_{n\in\mathbb{Z}} \left(\frac{4}{\pi^2}\left(\frac{\delta}{2}\right)^2 \int_{y_n}^{y_{n+1}} |f'(x)|^2 \, dx \right) = \frac{\delta^2}{\pi^2} \sum_{n\in\mathbb{Z}} \int_{y_n}^{y_{n+1}} |f'(x)|^2 \, dx \\
&= \frac{\delta^2}{\pi^2} \|f'\|^2 .
\end{aligned}$$

By Theorem 4.1.2 it holds that

$$\|f - Af\|^2 \leq \frac{\delta^2}{\pi^2} \|f'\|^2 \leq \frac{\delta^2}{\pi^2} \left(2\pi \sqrt{M_{\phi,h}} \|f\| \right)^2 = \left(2\delta \sqrt{M_{\phi,h}} \|f\| \right)^2,$$

which shows (4.2). We can now use (4.2) to show (4.3) as

$$\|Af\| = \|Af - f + f\| \leq \|Af - f\| + \|f\| \leq 2\delta\sqrt{M_{\phi,h}} \|f\| + \|f\| = (1 + 2\delta\sqrt{M_{\phi,h}}) \|f\| .$$

Lastly, if $\delta < \frac{1}{2\sqrt{M_{\phi,h}}}$ then $\|I-A\|<1$. Thus, by Lemma 4.1.5 we get

$$\|A^{-1}\| \leq \frac{1}{1-\|I-A\|} \overset{(4.2)}{\leq} \frac{1}{1-2\delta\sqrt{M_{\phi,h}}},$$

and consequently

$$\|A^{-1}f\| \leq \|A^{-1}\|\,\|f\| \leq \frac{1}{1-2\delta\sqrt{M_{\phi,h}}}\,\|f\| = (1-2\delta\sqrt{M_{\phi,h}})^{-1}\,\|f\|\,.$$

This shows the last statement. □

We are now ready to give a proof for the sampling bounds.

Theorem 4.1.7 *Let $\phi \in \mathcal{A}$ be a stable generator for $V_h(\phi)$ and let $\{x_n\}_{n\in\mathbb{Z}}$ with $x_n \in \mathbb{R}$. If $\delta = \sup_{n\in\mathbb{Z}}(x_{n+1}-x_n) < \frac{1}{2\sqrt{M_{\phi,h}}}$, then for every $f \in V_h(\phi)$ it holds that*

$$(1-2\delta\sqrt{M_{\phi,h}})^2\|f\|^2 \leq \sum_{n\in\mathbb{Z}} w_n|f(x_n)|^2 \leq (1+2\delta\sqrt{M_{\phi,h}})^2\|f\|^2, \tag{4.5}$$

where $w_n = (x_{n+1}-x_{n-1})/2$.

Proof. Using (4.4) we have that:

$$\begin{aligned}\|f\|^2 = \|A^{-1}Af\|^2 &\leq (1-2\delta\sqrt{M_{\phi,h}})^{-2}\|Af\|^2\\ &= (1-2\delta\sqrt{M_{\phi,h}})^{-2}\left\|P\left(\sum_{n\in\mathbb{Z}} f(x_n)\chi_{[y_n,y_{n+1}]}\right)\right\|^2\\ &\leq (1-2\delta\sqrt{M_{\phi,h}})^{-2}\left\|\sum_{n\in\mathbb{Z}} f(x_n)\chi_{[y_n,y_{n+1}]}\right\|^2.\end{aligned}$$

The last inequality follows from the fact that P is an orthogonal projection, and thus $\|Pf\| \leq \|P\|\,\|f\| = \|f\|$ by Lemma 2.1.3.

Observe that

$$y_{n+1}-y_n = \frac{x_n+x_{n+1}}{2} - \frac{x_{n-1}+x_n}{2} = \frac{x_{n+1}-x_{n-1}}{2} = w_n$$

then by using the fact that the intervals $[y_n, y_{n+1}]$ are non-overlapping, we get:

$$\begin{aligned}\left\|\sum_{k\in\mathbb{Z}} f(x_k)\chi_{[y_k,y_{k+1}]}\right\|^2 &= \int_{-\infty}^{\infty}\left|\sum_{k\in\mathbb{Z}} f(x_k)\chi_{[y_k,y_{k+1}]}\right|^2\,dx\\ &= \sum_{n\in\mathbb{Z}}\int_{y_n}^{y_{n+1}}\left|\sum_{k\in\mathbb{Z}} f(x_k)\chi_{[y_k,y_{k+1}]}\right|^2\,dx\\ &= \sum_{n\in\mathbb{Z}}\int_{y_n}^{y_{n+1}}|f(x_n)|^2\,dx\\ &= \sum_{n\in\mathbb{Z}}(y_{n+1}-y_n)\,|f(x_n)|^2\\ &= \sum_{n\in\mathbb{Z}} w_n\,|f(x_n)|^2\,. \end{aligned} \tag{4.6}$$

This shows that

$$\|f\|^2 \leq (1 - 2\delta\sqrt{M_{\phi,h}})^{-2} \left\| \sum_{n\in\mathbb{Z}} f(x_n)\chi_{[y_n,y_{n+1}]} \right\|^2 = (1 - 2\delta\sqrt{M_{\phi,h}})^{-2} \sum_{n\in\mathbb{Z}} w_n \, |f(x_n)|^2 .$$

And thus

$$(1 - 2\delta\sqrt{M_{\phi,h}})^2 \, \|f\|^2 \leq \sum_{n\in\mathbb{Z}} w_n |f(x_n)|^2.$$

This shows the lower bound of the inequality (4.5). By using Eq. (4.6) and the reverse triangle equality, we get:

$$\begin{aligned}
\sqrt{\sum_{n\in\mathbb{Z}} w_n|f(x_n)|^2} - \|f\| &= \left\| \sum_{n\in\mathbb{Z}} f(x_n)\chi_{[y_n,y_{n+1}]} \right\| - \|f\| \\
&\leq \left\| \sum_{n\in\mathbb{Z}} f(x_n)\chi_{[y_n,y_{n+1}]} - f \right\| \\
&= \left\| \sum_{n\in\mathbb{Z}} f(x_n)\chi_{[y_n,y_{n+1}]} - \sum_{n\in\mathbb{Z}} f\chi_{[y_n,y_{n+1}]} \right\| \\
&= \left\| \sum_{n\in\mathbb{Z}} (f - f(x_n))\chi_{[y_n,y_{n+1}]} \right\| .
\end{aligned}$$

Which implies that

$$\sum_{n\in\mathbb{Z}} w_n|f(x_n)|^2 \leq \left(\left\| \sum_{n\in\mathbb{Z}} (f - f(x_n))\chi_{[y_n,y_{n+1}]} \right\| + \|f\| \right)^2 .$$

By the proof of Theorem 4.1.6 we know that

$$\|f - Af\|^2 \leq \left\| \sum_{n\in\mathbb{Z}} (f - f(x_n))\chi_{[y_n,y_{n+1}]} \right\|^2 \leq \left(2\delta\sqrt{M_{\phi,h}} \, \|f\| \right)^2 .$$

Thus,

$$\begin{aligned}
\sum_{n\in\mathbb{Z}} w_n|f(x_n)|^2 &\leq \left(\|f\| + \left\| \sum_{n\in\mathbb{Z}} (f - f(x_n))\chi_{[y_n,y_{n+1}]} \right\| \right)^2 \\
&\leq \left(\|f\| + 2\delta\sqrt{M_{\phi,h}} \, \|f\| \right)^2 \\
&= \left(1 + 2\delta\sqrt{M_{\phi,h}} \right)^2 \|f\|^2 .
\end{aligned}$$

which shows the upper bound in the inequality (4.5). □

Corollary 4.1.8 *Let $\phi \in \mathcal{A}$ be a stable generator for $V_{1/\beta}(\phi)$. If $0 < \alpha < \frac{1}{2\sqrt{M_{\phi,1/\beta}}}$, then the set $x + \alpha\mathbb{Z} = \{x + n\alpha \mid n \in \mathbb{Z}\}$ is a set of stable sampling for $V_{1/\beta}(\phi)$ for $x \in \mathbb{R}$.*

Proof. Define $x_n = x + n\alpha$ and $h = \frac{1}{\beta}$. Then $\delta = \alpha$ in Theorem 4.1.7, and all the criteria for the theorem are met. Then, Eq. (4.5) in Theorem 4.1.7 becomes

$$(1 - 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \|f\|^2 \leq \sum_{n\in\mathbb{Z}} \alpha|f(x+n\alpha)|^2 \leq (1 + 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \|f\|^2,$$

or equivalently

$$\frac{1}{\alpha}(1 - 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \|f\|^2 \leq \sum_{n\in\mathbb{Z}} |f(x+n\alpha)|^2 \leq \frac{1}{\alpha}(1 + 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \|f\|^2.$$

We see from Eq. (3.5) that $x + \alpha\mathbb{Z}$ is a stable sampling set for $V_{1/\beta}(\phi)$. □

This leads us to the following theorem.

Theorem 4.1.9 *Let $\phi \in \mathcal{A}$ be a stable generator for $V_{1/\beta}(\phi)$. If $0 < \alpha < \frac{1}{2\sqrt{M_{\phi,1/\beta}}}$, then $\mathcal{G}(\phi, \alpha, \beta)$ forms a frame from $L^2(\mathbb{R})$ with frame bounds:*

$$A = \frac{1}{\alpha}(1 - 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \operatorname*{ess\,inf}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{\phi}(w+n\beta)|^2, \tag{4.7}$$

$$B = \frac{1}{\alpha}(1 + 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \operatorname*{ess\,sup}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{\phi}(w+n\beta)|^2. \tag{4.8}$$

Proof. From Theorem 3.4.7, we know that

$$A = C \operatorname*{ess\,inf}_{w\in[0,\beta]} \left(\sum_{n\in\mathbb{Z}} |\widehat{\phi}(w+n\beta)|^2\right) > 0, \quad B = D \operatorname*{ess\,sup}_{w\in[0,\beta]} \left(\sum_{n\in\mathbb{Z}} |\widehat{\phi}(w+n\beta)|^2\right) > 0.$$

Where C and D are the lower and upper frame bounds. Corollary 4.1.8 shows that

$$C = \frac{1}{\alpha}(1 - 2\alpha\sqrt{M_{\phi,h}})^2 \quad \text{and} \quad D = \frac{1}{\alpha}(1 + 2\alpha\sqrt{M_{\phi,h}})^2.$$

Consequently, it follows that:

$$A = \frac{1}{\alpha}(1 - 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \operatorname*{ess\,inf}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{\phi}(w+n\beta)|^2,$$

$$B = \frac{1}{\alpha}(1 + 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \operatorname*{ess\,sup}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{\phi}(w+n\beta)|^2.$$

□

4.2 The B-spline specific case

In this section, we will study using the B-spline Q_m as the window function ϕ for the Ghosh-Selvan method shown above. The B-spline and some of its properties were defined earlier in Definition 2.3.1 and Proposition 2.3.3 respectively. We will now show how we can rewrite $B_{Q_m,1/\beta}(w)$ from Definition 4.1.1 to a close form expression when using B-splines Q_m.

To use the Ghosh-Selvan method, we first need to show that $Q_m \in \mathcal{A}$. Since B-splines are continuous, bounded, and have compact support it is clear that (i) and (ii) hold in Definition 4.1.1. To show (iii) we need Theorem 4.2.4, which will be shown and proven later in this section.

Recall from Lemma 3.4.3 that Q_m is a stable generator for $V_{1/\beta}(Q_m)$ if and only if

$$0 < \beta \operatorname*{ess\,inf}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{Q}_m(w+n\beta)|^2 \le \beta \operatorname*{ess\,sup}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{Q}_m(w+n\beta)|^2 < \infty.$$

To show when Q_m is a stable generator, we consider the two cases: β is a positive integer and β is positive but not an integer. We start by considering the case where β is a positive integer. Then

$$|\widehat{Q}_m(w+n\beta)|^2 = \left|\left(\frac{\sin(\pi(w+n\beta))}{\pi(w+n\beta)}\right)^m\right|^2 = \left(\frac{\sin(\pi w)}{\pi(w+n\beta)}\right)^{2m}.$$

If $\beta = 1$, $w = 1$, and $n = -1$, we get $\left(\frac{\sin(\pi(1-1))}{\pi(1-1)}\right)^{2m} = \operatorname{sinc}(0)^{2m} = 1$. However, this only works for $\beta = 1$, as we are otherwise able to choose an integer $w < \beta$ such that $w+n\beta \neq 0$ for $n \in \mathbb{Z}$. Then $\left(\frac{\sin(\pi(w+n\beta))}{\pi(w+n\beta)}\right)^{2m} = 0$ since $\sin(\pi(w+n\beta)) = \sin(\pi w) = 0$. This shows that

$$\beta \operatorname*{ess\,inf}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{Q}_m(w+n\beta)|^2 = 0$$

for $\beta \in \{2, 3, \ldots\}$. We now consider the case where β is not an integer. Then $\sin(\pi w + \pi n\beta) > 0$ for some $n \in \mathbb{Z}$, so $\beta \operatorname{ess\,inf}_{w\in[0,\beta]} \sum_{n\in\mathbb{Z}} |\widehat{Q}_m(w+n\beta)|^2 > 0$. Furthermore, for $w \in [0, \beta]$:

$$\sum_{n\in\mathbb{Z}} \left|\left(\frac{\sin(\pi(w+n\beta))}{\pi(w+n\beta)}\right)^m\right|^2 \le \operatorname{sinc}(w)^{2m} + \sum_{n\in\mathbb{Z}\setminus\{0\}} \left(\frac{1}{\pi(w+n\beta}\right)^{2m} < \infty.$$

Consequently, we know that Q_m is a stable generator for $V_{1/\beta}(Q_m)$ except when β is an integer greater than one.

We define a new space called the Wiener space $W(\mathbb{R})$, which is needed to use the Poisson summation formula given as a lemma later in this section.

Definition 4.2.1 (The Wiener space) *The Wiener space is defined as*

$$W(\mathbb{R}) := \left\{ f \text{ is continuous on } \mathbb{R} \;\middle|\; \sum_{n\in\mathbb{Z}} \max_{x\in[0,1]} |f(x+n)| < \infty \right\}.$$

We want to show that the B-spline and its Fourier transformation is in the Wiener space.

Lemma 4.2.2 *Let $m \ge 2$, then Q_m and $\widehat{Q}_m$ belongs to $W(\mathbb{R})$.*

Proof. By Proposition 2.3.3 we know that Q_m is continuous and has compact support thus $Q_m \in W(\mathbb{R})$. We also know that $\widehat{Q}_m(w) = \left(\frac{\sin(\pi w)}{\pi w}\right)^m$ by Proposition 2.3.3. Therefore,

$$\sum_{n\in\mathbb{Z}} \max_{x\in[0,1]} |\widehat{Q}_m(x+n)| = \sum_{n\in\mathbb{Z}} \max_{x\in[0,1]} \left(\frac{\sin(\pi(x+n))}{\pi(x+n)}\right)^m \le 1 + 2\sum_{n=1}^{\infty} \left(\frac{1}{\pi n}\right)^m < \infty$$

since $\operatorname{sinc}(0) = 1$ and $m \geq 2$. This shows that $\widehat{Q}_m(w) \in W(\mathbb{R})$. □

The Poisson summation formula is given in the following lemma.

Lemma 4.2.3 (Poisson summation formula) *[14] If ϕ and $\widehat{\phi}$ belongs to $W(\mathbb{R})$, then the Poisson summation formula*

$$\sum_{n\in Z} \phi(x+\nu n) = \frac{1}{\nu}\sum_{n\in\mathbb{Z}} \widehat{\phi}\left(\frac{n}{\nu}\right) e^{2\pi i n x/\nu}, \quad \nu > 0$$

holds for all $x \in \mathbb{R}$ with absolute convergence of both sums.

We want a way to compute $B_{Q_m,1/\beta}(w)$ by rewriting the two sums to use Q_m instead of $\widehat{Q}_m$ since we know that Q_m has compact support, which means that we can compute $B_{Q_m,1/\beta}(w)$ by evaluating two finite sums.

Theorem 4.2.4 *Let Q_m be a B-spline of order m, $y = 2\pi w/\beta$, and $c = Q_{2m-2}(0) - Q_{2m-2}(1)$ then*

$$\sum_{n\in\mathbb{Z}} |\widehat{Q}_m(w+n\beta)|^2 = \frac{1}{\beta}\left(Q_{2m}(0) + 2\sum_{n=1}^{\lceil m\beta\rceil} Q_{2m}\left(\frac{n}{\beta}\right)\cos(ny)\right)$$

and

$$\sum_{n\in\mathbb{Z}} (w+n\beta)^2 |\widehat{Q}_m(w+n\beta)|^2$$
$$= \frac{1}{2\pi^2\beta}\left(c + \sum_{n=1}^{\lceil m\beta\rceil}\left(2Q_{2m-2}\left(\frac{n}{\beta}\right) - Q_{2m-2}\left(\frac{n}{\beta}+1\right) - Q_{2m-2}\left(\frac{n}{\beta}-1\right)\right)\cos(ny)\right).$$

Consequently, we have that

$$B_{Q_m,1/\beta}(w) = \frac{1}{2\pi^2}\frac{c + \sum_{n=1}^{\lceil m\beta\rceil}\left(2Q_{2m-2}\left(\frac{n}{\beta}\right) - Q_{2m-2}\left(\frac{n}{\beta}+1\right) - Q_{2m-2}\left(\frac{n}{\beta}-1\right)\right)\cos(ny)}{Q_{2m}(0) + 2\sum_{n=1}^{\lceil m\beta\rceil} Q_{2m}\left(\frac{n}{\beta}\right)\cos(ny)}.$$

Proof. First note by Proposition 2.3.3 for $\gamma \in \mathbb{R}$ that

$$|\widehat{Q}_m(\gamma)|^2 = \left|\left(\frac{\sin(\pi\gamma)}{\pi\gamma}\right)^m\right|^2 = \left(\frac{\sin(\pi\gamma)}{\pi\gamma}\right)^{2m} = \widehat{Q}_{2m}(\gamma). \tag{4.9}$$

By Lemma 4.2.2, Lemma 4.2.3 (Poisson summation formula), and Theorem 2.2.2 we can compute

$$\sum_{n\in\mathbb{Z}} |\widehat{Q}_m(w+n\beta)|^2 \overset{(4.9)}{=} \sum_{n\in\mathbb{Z}} \widehat{Q}_{2m}(w+n\beta) = \frac{1}{\beta}\sum_{n\in\mathbb{Z}} \widehat{\widehat{Q}}_{2m}\left(\frac{n}{\beta}\right) e^{2\pi i n w/\beta}$$
$$= \frac{1}{\beta}\sum_{n\in\mathbb{Z}} Q_{2m}\left(-\frac{n}{\beta}\right) e^{2\pi i n w/\beta} = \frac{1}{\beta}\sum_{n\in\mathbb{Z}} Q_{2m}\left(\frac{n}{\beta}\right) e^{2\pi i n w/\beta}.$$

Let $y = 2\pi w/\beta$ then by Proposition 2.3.3 we know that $\operatorname{supp} Q_m = [-\frac{m}{2}, \frac{m}{2}]$ thus

$$\begin{aligned}\sum_{n\in\mathbb{Z}} |\widehat{Q}_m(w+n\beta)|^2 &= \frac{1}{\beta}\sum_{n\in\mathbb{Z}} Q_{2m}\left(\frac{n}{\beta}\right)e^{2\pi i n w/\beta} = \frac{1}{\beta}\sum_{n=-\lceil m\beta\rceil}^{\lceil m\beta\rceil} Q_{2m}\left(\frac{n}{\beta}\right)e^{2\pi i n w/\beta}\\ &= \frac{1}{\beta}\left(Q_{2m}(0) + 2\sum_{n=1}^{\lceil m\beta\rceil} Q_{2m}\left(\frac{n}{\beta}\right)\cos(yn)\right)\end{aligned}$$

since Q_m is an even function and $e^{2\pi i n w/\beta} = \cos(2\pi n w/\beta) + i\sin(2\pi n w/\beta)$, where $\sin(x)$ is an odd function and $\cos(x)$ is an even function.

Note that by Theorem 2.2.4 we have for any $f \in L^1(\mathbb{R})$ and $\gamma \in \mathbb{R}$ then

$$\widehat{f''}(\gamma) = 2\pi i\gamma\widehat{f'}(\gamma) = (2\pi i\gamma)^2\widehat{f}(\gamma) = -4\pi^2\gamma^2\widehat{f}(\gamma) \tag{4.10}$$

thus we get

$$\gamma^2|\widehat{Q}_m(\gamma)|^2 \overset{(4.9)}{=} \gamma^2\widehat{Q}_{2m}(\gamma) = -\frac{1}{4\pi^2}(-4\pi^2\gamma^2)\widehat{Q}_{2m}(\gamma) \overset{(4.10)}{=} -\frac{1}{4\pi^2}\widehat{Q''}_{2m}(\gamma). \tag{4.11}$$

By Lemma 4.2.2, Lemma 4.2.3 (Poisson summation formula), and Theorem 2.2.2 we can compute

$$\begin{aligned}\sum_{n\in\mathbb{Z}}(w+n\beta)^2|\widehat{Q}_m(w+n\beta)|^2 &\overset{(4.11)}{=} -\frac{1}{4\pi^2}\sum_{n\in\mathbb{Z}}\widehat{Q''}_{2m}(w+n\beta)\\ &= -\frac{1}{4\pi^2\beta}\sum_{n\in\mathbb{Z}}\widehat{\widehat{Q''}}_{2m}\left(\frac{n}{\beta}\right)e^{2\pi i n w/\beta} = -\frac{1}{4\pi^2\beta}\sum_{n\in\mathbb{Z}}Q''_{2m}\left(\frac{n}{\beta}\right)e^{2\pi i n w/\beta}.\end{aligned}$$

Let $y = 2\pi w/\beta$ and $c = Q_{2m-2}(0) - Q_{2m-2}(1)$ then by Proposition 2.3.3 we get

$$\begin{aligned}&\sum_{n\in\mathbb{Z}}(w+n\beta)^2|\widehat{Q}_m(w+n\beta)|^2 = -\frac{1}{4\pi^2\beta}\sum_{n\in\mathbb{Z}}Q''_{2m}\left(\frac{n}{\beta}\right)e^{2\pi i n w/\beta}\\ &= -\frac{1}{4\pi^2\beta}\sum_{n\in\mathbb{Z}}\left(Q'_{2m-1}\left(\frac{n}{\beta}+\frac{1}{2}\right) - Q'_{2m-1}\left(\frac{n}{\beta}-\frac{1}{2}\right)\right)e^{2\pi i n w/\beta}\\ &= -\frac{1}{4\pi^2\beta}\sum_{n\in\mathbb{Z}}\left(Q_{2m-2}\left(\frac{n}{\beta}+1\right) + Q_{2m-2}\left(\frac{n}{\beta}-1\right) - 2Q_{2m-2}\left(\frac{n}{\beta}\right)\right)e^{2\pi i n w/\beta}\\ &= \frac{1}{4\pi^2\beta}\sum_{n=-\lceil m\beta\rceil}^{\lceil m\beta\rceil}\left(2Q_{2m-2}\left(\frac{n}{\beta}\right) - Q_{2m-2}\left(\frac{n}{\beta}+1\right) - Q_{2m-2}\left(\frac{n}{\beta}-1\right)\right)e^{2\pi i n w/\beta}\\ &= \frac{1}{2\pi^2\beta}\left(c + \sum_{n=1}^{\lceil m\beta\rceil}\left(2Q_{2m-2}\left(\frac{n}{\beta}\right) - Q_{2m-2}\left(\frac{n}{\beta}+1\right) - Q_{2m-2}\left(\frac{n}{\beta}-1\right)\right)\cos(ny)\right)\end{aligned}$$

since Q_m is an even function and $e^{2\pi i n w/\beta} = \cos(2\pi n w/\beta) + i\sin(2\pi n w/\beta)$, where $\sin(x)$ is an odd function and $\cos(x)$ is an even function. □

In Theorem 4.2.4 we give a way to compute $B_{Q_m,1/\beta}(w)$ without using the Fourier transform. It now becomes clear that (iii) in Definition 4.1.1 holds, as $\sum_{n\in\mathbb{Z}}(w+n\beta)^2|\widehat{Q}_m(w+n\beta)|^2 < \infty$ by Theorem 4.2.4.

Ghosh and Selvan used Theorem 4.2.4 to prove that $B_{Q_2,1/\beta}(w)$ attains it maximum at $w = \frac{\beta}{2}$, for $0 < \beta < 2$ [13, p. 9], hence they were able to avoid any discretization errors. As this is only shown for $\beta < 2$ and $m \in \{2,3\}$, we have decided not to implement this. We have estimated the frame region of Q_2 numerically using the method described above.

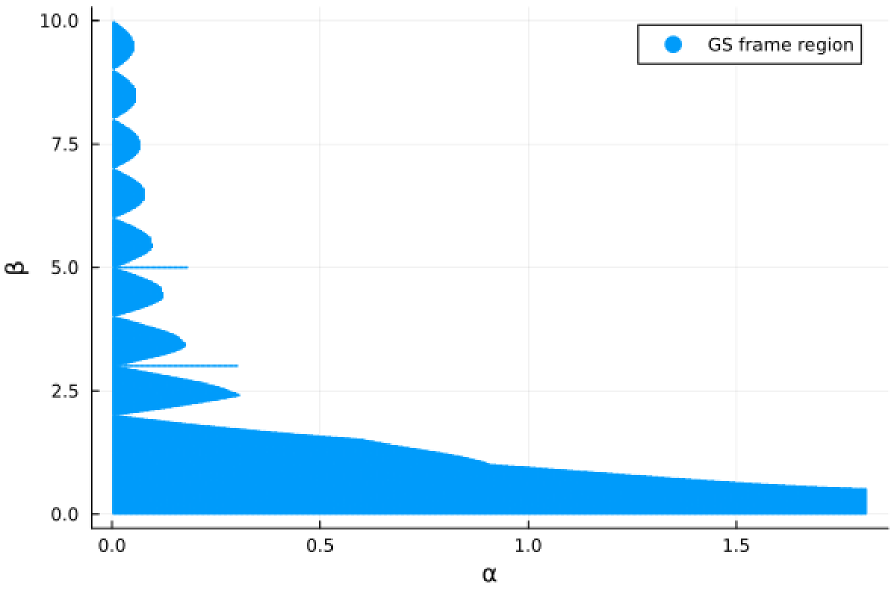

Figure 4.2: The computed frame set for Q_2. The spikes are due to numerical error and are not part of the frame set. The plot is generated in **scripts/gs_bspline_case.jl**.

We see that the numerical method have problems estimating the frame region for Q_2 when β is an integer, which makes sense as Q_m is not a stable generator on integers as shown in the beginning of this section.

Chapter 5

Theoretical bounds on Ghosh-Selvan for B-splines

As the method presented in Chapter 4 is not in a closed form we will use the following chapter to find a theoretical upper bound $M^*_{Q_m,1/\beta}$ for $M_{Q_m,1/\beta}$, such that we can find

$$\alpha_{\max} := \frac{1}{2\sqrt{M_{Q_m,1/\beta}}} \geq \frac{1}{2\sqrt{M^*_{Q_m,1/\beta}}}.$$

We know by Theorem 4.1.9 that $(\alpha, \beta) \in \mathfrak{F}(Q_m)$ for all $\alpha \in]0, \alpha_{\max}[$ for $m \geq 2$.

A key result in this chapter is Theorem 5.1.1 in which we show that for any positive non-integer β, we can find a corresponding α such that $(\alpha, \beta) \in \mathfrak{F}(Q_m)$.

We denote the numerator of $B_{Q_m,1/\beta}(w)$ as $\Omega(w)$ and the denominator as $\Xi(w)$. We can then find an upper bound on the supremum of Ω and a lower bound on the infimum of Ξ to obtain $M^*_{Q_m,1/\beta}$.

5.1 A simple bound

5.1.1 A simple upper bound on Ω

Using Theorem 4.2.4 with $y = 2\pi w/\beta$ and $c = Q_{2m-2}(0) - Q_{2m-2}(1)$ and the fact that $\operatorname{Im} Q_{2m-2} \subseteq [0, 1]$, we get that:

$$\begin{aligned}
\Omega(w) &= \sum_{n\in\mathbb{Z}} (w + n\beta)^2 |\widehat{Q}_m(w + n\beta)|^2 \\
&= \frac{1}{2\pi^2\beta}\left(c + \sum_{n=1}^{\lfloor m\beta\rfloor}\left(2Q_{2m-2}\left(\frac{n}{\beta}\right) - Q_{2m-2}\left(\frac{n}{\beta}+1\right) - Q_{2m-2}\left(\frac{n}{\beta}-1\right)\right)\cos(ny)\right) \\
&\leq \frac{1}{2\pi^2\beta}\left(1 + \sum_{n=1}^{\lfloor m\beta\rfloor}\left|2Q_{2m-2}\left(\frac{n}{\beta}\right) - Q_{2m-2}\left(\frac{n}{\beta}+1\right) - Q_{2m-2}\left(\frac{n}{\beta}-1\right)\right|\right) \\
&\leq \frac{1}{2\pi^2\beta}\left(1 + \lfloor m\beta\rfloor 2\right) \leq \frac{1 + 2m\beta}{2\pi^2\beta} := \Omega_0. \qquad (5.1)
\end{aligned}$$

5.1.2 A simple lower bound on Ξ

We seek a lower bound for the denominator

$$\sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w+\ell\beta)|^2.$$

By inserting $w+\beta$, we show that the Ξ is β-periodic

$$\sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m((w+\beta)+\ell\beta)|^2 = \sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w+(\ell+1)\beta)|^2 = \sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w+\ell\beta)|^2.$$

As $\widehat{Q}_m$ is even we also have that Ξ is an evenfunction

$$\sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(-w+\ell\beta)|^2 = \sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w-\ell\beta)|^2 = \sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w+\ell\beta)|^2.$$

Consequently, when taking the infimum, we get:

$$\begin{aligned}\inf_{w\in[0,\beta]} \sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w+\ell\beta)|^2 &= \inf_{w\in[-\beta/2,\beta/2]} \sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w+\ell\beta)|^2 \\ &= \inf_{w\in[0,\beta/2]} \sum_{\ell\in\mathbb{Z}} |\widehat{Q}_m(w+\ell\beta)|^2 \\ &\geq \inf_{w\in[0,\beta/2]} |\widehat{Q}_m(w)|^2 + |\widehat{Q}_m(w-\beta)|^2 \\ &= \inf_{w\in[0,\beta/2]} ((\operatorname{sinc}(w))^{2m} + (\operatorname{sinc}(w-\beta))^{2m}) := \Xi_0.\end{aligned}$$

As sinc has roots $\mathbb{Z}\backslash\{0\}$, the right-hand side of the inequality is positive whenever β is not an integer. Thus, we find the upper bound on $M_{Q_m,1/\beta}$

$$M_{Q_m,1/\beta} \leq \frac{\Omega_0}{\Xi_0} = \frac{1+2m\beta}{2\pi^2\beta} \frac{1}{\inf_{w\in[0,\beta/2]}((\operatorname{sinc}(w))^{2m} + (\operatorname{sinc}(w-\beta))^{2m})}, \tag{5.2}$$

which is finite for non-integer β, giving us the following theorem.

Theorem 5.1.1 *For every B-spline of order $m \geq 2$, and for every postive non-integer β, there exists an $\alpha_{\max} > 0$ such that for all $\alpha \in]0, \alpha_{\max}[$, we have that $(\alpha, \beta) \in \mathfrak{F}(Q_m)$.*

To find a lower bound of this $\alpha_{\max}$ we need a closed-form expression. We find one for rational values of β.

5.2 Closed form bounds for rational β

Now assume that $\beta = \frac{p}{q} \in \mathbb{Q}$ with $\gcd(p,q) = 1$, and $m, q \geq 2$. Then we have that:

$$\begin{aligned}\sin(\pi(w+(\ell+q)\beta))^{2m} &= \sin\left(\pi\left(w+(\ell+q)\frac{p}{q}\right)\right)^{2m} = \sin\left(\pi\left(w+\ell\frac{p}{q}+p\right)\right)^{2m} \\ &= \sin\left(\pi\left(w+\ell\frac{p}{q}\right)+p\pi\right)^{2m} = \sin\left(\pi\left(w+\ell\frac{p}{q}\right)\right)^{2m} \\ &= \sin(\pi(w+\ell\beta))^{2m}.\end{aligned}$$

Using the periodicity of $\sin(\pi(w+\ell\beta))^{2m}$ we define

$$\sigma_{q,m}(w) := \sum_{\ell=0}^{q-1} \sin(\pi(w+\ell\beta))^{2m} = \sum_{l=k}^{q-1+k} \sin(\pi(w+\ell\beta))^{2m}, \tag{5.3}$$

for any $k \in \mathbb{Z}$. Later we will show why we need $\sigma_{q,m}$. Before that, we need to know a few key details of $\sigma_{q,m}$. To do that we need the following identity.

Lemma 5.2.1 *Let $p, z \in \mathbb{Z}$, $q \in \mathbb{Z} \setminus \{0\}$, $w \in \mathbb{R}$, and $\gcd(p,q) = 1$. Then*

$$\sum_{\ell=0}^{q-1} \cos\left(2\pi z \left(w + \ell\frac{p}{q}\right)\right) = \begin{cases} 0 & q \nmid z \\ q\cos(2\pi z w) & q \mid z. \end{cases}$$

Proof. Note that by using Euler's equation $e^{ix} = \cos x + i \sin x$ for $x \in \mathbb{R}$, we get

$$\begin{aligned}\sum_{\ell=0}^{q-1} \cos\left(2\pi z \left(w + \ell\frac{p}{q}\right)\right) &= \frac{1}{2}\sum_{\ell=0}^{q-1}\left(e^{i2\pi z(w+\ell\frac{p}{q})} + e^{-i2\pi z(w+\ell\frac{p}{q})}\right) \\ &= \frac{1}{2}e^{i2\pi z w}\sum_{\ell=0}^{q-1}(e^{i2\pi z\frac{p}{q}})^{\ell} + \frac{1}{2}e^{-i2\pi z w}\sum_{\ell=0}^{q-1}(e^{-i2\pi z\frac{p}{q}})^{\ell} =: (*).\end{aligned}$$

If $q \nmid z$, then the following fractions (gained by using the geometric sum formula), are well defined as $\frac{zp}{q}$ is not an integer. Thus

$$(*) = \frac{1}{2}e^{i2\pi z w}\frac{1-(e^{i2\pi z\frac{p}{q}})^q}{1-e^{i2\pi z\frac{p}{q}}} + \frac{1}{2}e^{-i2\pi z w}\frac{1-(e^{-i2\pi z\frac{p}{q}})^q}{1-e^{-i2\pi z\frac{p}{q}}} = 0.$$

If $q \mid z$ then $\frac{zp}{q}$ is an integer, and $e^{i2\pi z\frac{p}{q}} = 1$, hence

$$(*) = \frac{q}{2}(e^{i2\pi z w} + e^{-i2\pi z w}) = q\cos(2\pi z w). \qquad \square$$

For the proof of the next lemma, we will need the power reduction formula for $\sin(\theta)^{2n}$, the proof of which follows by induction.

Lemma 5.2.2 (Power reduction formula) *Let $n \in \mathbb{Z}_{>0}$ then*

$$\sin(\theta)^{2n} = \frac{1}{4^n}\binom{2n}{n} + \frac{2}{4^n}\sum_{k=0}^{n-1}(-1)^{n-k}\binom{2n}{k}\cos(2\theta(n-k)).$$

Now we can show a few key facts about $\sigma_{q,m}$.

Proposition 5.2.3 *The function $\sigma_{q,m}$ is a $\frac{1}{q}$ periodic, even function. The value of $\sigma_{q,m}$ is independent of the value of p as long as $\gcd(p,q) = 1$, and $\sigma_{q,m}$ is constant when $m < q$. Furthermore, $\sigma_{q,m}$ can be written as*

$$\sigma_{q,m}(w) = \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor\frac{m}{q}\rfloor}(-1)^{q\zeta}\binom{2m}{m+q\zeta}\cos(2\pi q\zeta w). \tag{5.4}$$

Proof. First, note that by using the power reduction formula from Lemma 5.2.2 we get:

$$\begin{aligned}\sin(\pi(w+\ell\beta))^{2m} &= \frac{1}{4^m}\binom{2m}{m} + \frac{2}{4^m}\sum_{k=0}^{m-1}(-1)^{m-k}\binom{2m}{k}\cos(2(m-k)\pi(w+\ell\frac{p}{q}))\\ &= \frac{1}{4^m}\binom{2m}{m} + \frac{2}{4^m}\sum_{z=1}^{m}(-1)^{z}\binom{2m}{m+z}\cos(2z\pi(w+\ell\frac{p}{q})).\end{aligned}$$

Thus, using Lemma 5.2.1 it holds that:

$$\begin{aligned}\sigma_{q,m}(w) &= \frac{q}{4^m}\binom{2m}{m} + \frac{2}{4^m}\sum_{z=1}^{m}(-1)^{z}\binom{2m}{m+z}\sum_{\ell=0}^{q-1}\cos(2z\pi(w+\ell\frac{p}{q}))\\ &= \frac{q}{4^m}\binom{2m}{m} + \frac{2}{4^m}\sum_{\zeta=1}^{\lfloor\frac{m}{q}\rfloor}(-1)^{q\zeta}\binom{2m}{m+q\zeta}q\cos(2\pi q\zeta w)\\ &= \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor\frac{m}{q}\rfloor}(-1)^{q\zeta}\binom{2m}{m+q\zeta}\cos(2\pi q\zeta w).\end{aligned}$$

This shows that $\sigma_{q,m}$ is a $\frac{1}{q}$-periodic and even function, and that it is independent of p. Moreover, it shows that $\sigma_{q,m}$ is constant when $m < q$. $\square$

Lemma 5.2.4 $\sup_{w\in[0,\frac{p}{2q}]}\sigma_{q,m}(w) \geq 1$.

Proof. Note that there exists a $w \in [0, \frac{1}{q}[\subseteq [0, \frac{p}{q}]$ and $\ell \in \{1, \ldots, q\}$ such that $w+\ell\frac{1}{q} = \frac{1}{2}$, and thus we can always find a w such that one of the terms in $\sigma_{q,m}(w) = \sum_{\ell=1}^{q}\sin(\pi(w+\ell\frac{1}{q}))^{2m}$ is equal to 1. As $\sigma_{q,m}(w)$ is $\frac{1}{q}$-periodic and even, such a w exists in $[0, \frac{1}{2q}] \subseteq [0, \frac{p}{2q}]$. $\square$

To get an exact value of the supremum we have the following lemma.

Lemma 5.2.5 *The supremum of* $\sigma_{q,m}$ *is*

$$\sup_{w\in\mathbb{R}}\sigma_{q,m}(w) = \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor\frac{m}{q}\rfloor}\binom{2m}{m+q\zeta}.$$

Moreover, the supremum is achieved at $\frac{1}{2q}+\frac{1}{q}\mathbb{Z}$ *when* q *is odd, and at* $\frac{1}{q}\mathbb{Z}$ *when* q *is even, here* $\frac{1}{q}\mathbb{Z}$ *denotes the set* $\{\frac{n}{q} \mid n \in \mathbb{Z}\}$.

Proof. It is clear from Eq. (5.4) that $\sigma_{q,m}$ cannot take values larger than the stated supremum, thus we simply have to show that it takes this value at the stated points. As $\sigma_{q,m}$ is $\frac{1}{q}$-periodic, we only have to check the two basic cases. First, assume that

q is odd. Then

$$\begin{aligned}\sigma_{q,m}\left(\frac{1}{2q}\right) &= \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor \frac{m}{q} \rfloor}(-1)^{q\zeta}\binom{2m}{m+q\zeta}\cos\left(2\pi q\zeta\frac{1}{2q}\right)\\ &= \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor \frac{m}{q} \rfloor}(-1)^{\zeta}\binom{2m}{m+q\zeta}\cos\left(\pi\zeta\right)\\ &= \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor \frac{m}{q} \rfloor}\binom{2m}{m+q\zeta}.\end{aligned}$$

Now assume q is even. Then it holds that:

$$\begin{aligned}\sigma_{q,m}\left(0\right) &= \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor \frac{m}{q} \rfloor}\binom{2m}{m+q\zeta}\cos\left(2\pi q\zeta 0\right)\\ &= \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor \frac{m}{q} \rfloor}\binom{2m}{m+q\zeta}.\end{aligned}$$ □

Proposition 5.2.6 *Let $m, q \geq 2$, then $\inf_{w\in[0,\frac{p}{2q}]}\sigma_{q,m}(w)$ is positive.*

Proof. Using Proposition 5.2.3 we assume that $p = 1$. As each term in Eq. (5.3) is non-negative we simply have to show that the terms for $\ell = 0$ and $\ell = 1$ in Eq. (5.3) cannot be zero at the same time.

First, we consider the cases when the term for $\ell = 0$ is zero, i.e.

$$\sin\left(\pi\left(w + 0\frac{1}{q}\right)\right)^{2m} = \sin\left(\pi w\right)^{2m} = 0,$$

which happens only when $w \in \mathbb{Z}$. Considering the term for $\ell = 1$ with $w \in \mathbb{Z}$ we have $\sin\left(\pi\left(w + 1\frac{1}{q}\right)\right)^{2m} = \sin\left(\pi\left(w + \frac{1}{q}\right)\right)^{2m} \neq 0$ as $w + \frac{1}{q}$ is not an integer. Thus, as $\sigma_{q,m}$ is continuous we conclude that $\sigma_{q,m}(w) > 0$. □

We also state the following without proof:

Conjecture 5.2.7 *The infimum of $\sigma_{q,m}$ is attained at $\frac{1}{q}\mathbb{Z}$ when q is odd and at $\frac{1}{2q} + \frac{1}{q}\mathbb{Z}$ when q is even. Furthermore, the infimum is:*

$$\inf_{w\in[0,\frac{p}{2q}]}\sigma_{q,m}(w) = \frac{q}{4^m}\binom{2m}{m} + \frac{2q}{4^m}\sum_{\zeta=1}^{\lfloor \frac{m}{q} \rfloor}(-1)^{\zeta}\binom{2m}{m+q\zeta}.$$

This conjecture is trivially true when $q > m$ as $\lfloor \frac{m}{q} \rfloor = 0$. For $m \geq q > m/2$ this is also clearly true. For the case $q \leq m/2$, numeric simulations show that the conjecture holds, but as no proof has been found. Therefore, we will only show that these points are non supremum extremas, but not whether they are global minima.

First note that using Proposition 5.2.3:

$$\frac{d\sigma_{q,m}}{dw}(w) = -\frac{4\pi q^2}{4^m}\sum_{\zeta=1}^{\lfloor \frac{m}{q} \rfloor}(-1)^{q\zeta}\binom{2m}{\zeta q+m}\zeta\sin(2\pi q\zeta w)\,.$$

Thus, $\frac{d\sigma_{q,m}}{dw}(0) = \frac{d\sigma_{q,m}}{dw}\left(\frac{1}{2q}\right) = 0$. Hence, these points are extremas.

At these points for odd and even q respectively, $\sigma_{q,m}$ takes the value stated in the conjecture. For $q \le m$ this value of $\sigma_{q,m}$ is less than the supremum of $\sigma_{q,m}$ in Lemma 5.2.5, hence we know that these points cannot be maximas.

5.2.1 A closed form lower bound on Ξ

Now we are able to lower bound the denominator of $B_{Q_m,1/\beta}(w)$ on $[0, \frac{p}{2q}]$, where $\beta = \frac{p}{q}$ and $q \ge 2$.

$$\begin{aligned}
\Xi(w)\cdot\pi^{2m} &= \pi^{2m}\sum_{\ell\in\mathbb{Z}}|\widehat{Q}_m(w+\ell\beta)|^2 = \sum_{\ell\in\mathbb{Z}}(w+\ell\beta)^{-2m}\sin(\pi(w+\ell\beta))^{2m}\\
&= \sum_{\ell=1}^{\infty}\left(((\ell-1)\beta+w)^{-2m}\sin(\pi(w+(\ell-1)\beta))^{2m} + (\ell\beta-w)^{-2m}\sin(\pi(\ell\beta-w))^{2m}\right)\\
&= \sum_{\ell=0}^{\infty}\sum_{a=1}^{q}((\ell q+a-1)\beta+w)^{-2m}\sin(\pi(w+(\ell q+a-1)\beta))^{2m}\\
&\quad + \sum_{\ell=0}^{\infty}\sum_{a=1}^{q}((\ell q+a)\beta-w)^{-2m}\sin(\pi((\ell q+a)\beta-w))^{2m})\\
&> \sum_{\ell=0}^{\infty}\sum_{a=1}^{q}((\ell q+q-1)\beta+w)^{-2m}\sin(\pi(w+(\ell q+a-1)\beta))^{2m}\\
&\quad + \sum_{\ell=0}^{\infty}\sum_{a=1}^{q}((\ell q+q)\beta-w)^{-2m}\sin(\pi((\ell q+a)\beta-w))^{2m})\\
&= \sum_{\ell=0}^{\infty}((\ell q+q-1)\beta+w)^{-2m}\sigma_{q,m}(w) + \sum_{\ell=0}^{\infty}((\ell q+q)\beta-w)^{-2m}\sigma_{q,m}(w)\\
&= \sigma_{q,m}(w)\sum_{\ell=1}^{\infty}\left(((\ell q-1)\beta+w)^{-2m} + (\ell q\beta-w)^{-2m}\right)\\
&= \sigma_{q,m}(w)\sum_{\ell=1}^{\infty}\left(\left(\ell p-\frac{p}{q}+w\right)^{-2m} + (\ell p-w)^{-2m}\right) =: (*)
\end{aligned}$$

The terms of the sum are decreasing, thus we can use that $\sum_{i=1}^{\infty} f(i) \ge \int_1^{\infty} f(x)\,dx$, by Corollary 4.35 in [5, p. 103]. Thus

$$\begin{aligned}
(*) &\ge \sigma_{q,m}(w)\int_1^{\infty}\left(\left(\ell p-\frac{p}{q}+w\right)^{-2m} + (\ell p-w)^{-2m}\right)\,d\ell\\
&= \sigma_{q,m}(w)\frac{\left(p+w-\frac{p}{q}\right)^{1-2m} + (p-w)^{1-2m}}{(2m-1)\,p} =: \Xi_1(w)\cdot\pi^{2m}.
\end{aligned}$$

Lemma 5.2.8 *Denote the constant* $\nu_{q,m} := \frac{4^m}{q} \cdot \inf_{w\in[0,\frac{1}{2q}]} \sigma_{q,m}$. *Then*

$$\inf_{w\in[0,\frac{p}{2q}]} \Xi_1(w) \geq \nu_{q,m} \cdot \frac{p^{-2m} q^{2m} \left(2q-1\right)^{1-2m}}{\pi^{2m}(2m-1)}.$$

Proof. As $\sigma_{q,m}$ is even and $\frac{1}{q}$-periodic, we have that

$$\Xi_1(w) \geq \pi^{-2m} \inf_{w\in[0,\frac{1}{2q}]} \sigma_{q,m}\left(w\right) \cdot \frac{\inf_{w\in[0,\frac{p}{2q}]}\left(\left(p+w-\frac{p}{q}\right)^{1-2m} + \left(p-w\right)^{1-2m}\right)}{\left(2m-1\right)p}.$$

The last factor in $\Xi_1(w)$ is non-increasing for $w \in [0, \frac{p}{2q}]$, thus we can find its infimum at $\frac{p}{2q}$.

$$\begin{aligned}\Xi_1(w) &\geq \pi^{-2m} \inf_{w\in[0,\frac{1}{2q}]} \sigma_{q,m}(w) \cdot \frac{p^{-2m} 4^m q^{2m-1} \left(2q-1\right)^{1-2m}}{2m-1} \\ &= \nu_{q,m} \cdot \frac{p^{-2m} q^{2m} \left(2q-1\right)^{1-2m}}{\pi^{2m}(2m-1)}.\end{aligned}$$ □

This gives us the following theorem:

Theorem 5.2.9 *For* $m \geq 2$, $\beta = \frac{p}{q} \in \mathbb{Z}_{>0}$ *where* $\gcd(p,q) = 1$ *and* $q \geq 2$. *Then for*

$$\alpha \in \left]0, p^{-m}\frac{\pi}{\pi^m}\sqrt{\frac{\nu_{q,m} q^{2m} \left(2q-1\right)^{1-2m}}{2(\frac{q}{p}+2m)(2m-1)}}\right].$$

We have that $\mathcal{G}(Q_m, \alpha, \beta)$ *is a frame.*

Proof. Using Ω_0 as given in (5.1) as a denominator, Lemma 5.2.8 allows us to find a simple closed form upper bound on $M_{Q_m,1/\beta}$.

$$\begin{aligned}M_{Q_m,1/\beta} &< \frac{\frac{\frac{q}{p}+2m}{2\pi^2}}{\nu_{q,m} \cdot \frac{p^{-2m}q^{2m}(2q-1)^{1-2m}}{\pi^{2m}(2m-1)}} \\ &= \pi^{2m-2}\frac{(\frac{q}{p}+2m)(2m-1)}{\nu_{q,m} \cdot 2p^{-2m}q^{2m}\left(2q-1\right)^{1-2m}}.\end{aligned}$$

Which gives us the following:

$$\begin{aligned}\alpha_{\max} = \frac{1}{2\sqrt{M_{Q_m,1/\beta}}} &> \frac{1}{2}p^{-m}\frac{\pi}{\pi^m}\sqrt{\frac{\nu_{q,m}\cdot 2q^{2m}\left(2q-1\right)^{1-2m}}{(\frac{q}{p}+2m)(2m-1)}} \\ &= p^{-m}\frac{\pi}{\pi^m}\sqrt{\frac{\nu_{q,m}q^{2m}\left(2q-1\right)^{1-2m}}{2(\frac{q}{p}+2m)(2m-1)}}.\end{aligned}$$

This completes the proof. □

If we assume $\beta = \frac{p}{q} > 1$, then we can lower bound $\alpha_{\max}$ even further to make a bound that is easier to compute:

$$\begin{aligned}
\alpha_{\max} &> p^{-m}\frac{\pi}{\pi^m}\sqrt{\frac{\nu_{q,m}q^{2m}\left(2q-1\right)^{1-2m}}{2(1+2m)(2m-1)}} \\
&= p^{-m}\frac{\pi}{\pi^m}\sqrt{\frac{\nu_{q,m}q^{2m}\left(2q-1\right)^{1-2m}}{8m^2-2}} \\
&> p^{-m}\frac{\pi}{\pi^m 2^m}\sqrt{\frac{\nu_{q,m}(2q-1)^{2m}\left(2q-1\right)^{1-2m}}{8m^2}} \\
&= p^{-m}\frac{\pi}{(2\pi)^m m}\sqrt{\frac{\nu_{q,m}(2q-1)}{8}}. \qquad (5.5)
\end{aligned}$$

Note that $\nu_{q,m}$ is easy to calculate, when $q > m$ as by Proposition 5.2.3:

$$\nu_{q,m} = \frac{4^m}{q}\inf \sigma_{q,m} = \frac{4^m}{q}\frac{q}{4^m}\binom{2m}{m} = \binom{2m}{m}.$$

If we assume Conjecture 5.2.7 then for $q \le m$ it holds that

$$\nu_{q,m} = \binom{2m}{m} + 2\sum_{\zeta=1}^{\lfloor \frac{m}{q}\rfloor}(-1)^{q\zeta}\binom{2m}{m+q\zeta}\cos(2\pi q\zeta w) = \binom{2m}{m} + 2\sum_{\zeta=1}^{\lfloor \frac{m}{q}\rfloor}(-1)^{\zeta}\binom{2m}{m+q\zeta}.$$

Also note that (5.5) can be expressed as a constant depending only on q and m multiplied with p to some power, making it easy to compute for additional values of p for fixed q and m.

5.2.2 Finding a smaller upper bound of Ω

We can find upper bounds of $\Omega(w)$ that are smaller than Ω_0 as given in (5.1), but first we need the following inequality.

Lemma 5.2.10 *For $n \in \mathbb{Z}$ such that $n \ge 2$ and constants $0 \le a \le w \le b < k$, then*

$$(k-a)^{-n} + (k+a)^{-n} \le (k-w)^{-n} + (k+w)^{-n} \le (k-b)^{-n} + (k+b)^{-n}.$$

Proof. Note that for $w \in [0,k[$ then

$$\frac{d}{dx}\left[(k-x)^{-n} + (k+x)^{-n}\right](w) = n\left((k-w)^{-n-1} + (k+w)^{-n-1}\right) \ge 0.$$

Thus, $(k-x)^{-n} + (k+x)^{-n}$ is non-decreasing for $x \in [0,k[$ and the inequality follows. □

First, let $d \in]0, \frac{\beta}{2}]$ and consider $w \in [0,d]$. Then we have that

$$\sup_{w\in[0,d]} w^{2-2m}\sin(\pi w)^{2m} = \sup_{w\in[0,d]} w^2\pi^{2m}\operatorname{sinc}(w) \le \sup_{w\in[0,d]} w^2\pi^{2m} = d^2\pi^{2m}.$$

Using that $\sin(w) \leq 1$, we have that for $w \in [0, d]$

$$w^{2-2m} \sin(\pi w)^{2m} \leq w^{2-2m}.$$

The first bound is increasing as a function of d while the second one is decreasing as a function of w. The smallest upper bound on $\sup_{w\in[0,d]} w^{2-2m} \sin(\pi w)^{2m}$ is where these two bounds are equal. That is when

$$w^2 \pi^{2m} = w^{2-2m} \iff \pi^{2m} = w^{-2m} \iff w = \frac{1}{\pi}.$$

Thus, on $[0, d]$ we see that:

$$w^2 \sin(\pi w)^{2m} \leq \min(d^2 \pi^{2m}, \pi^{2m-2}) =: W.$$

Then it holds that

$$\begin{aligned}
\Omega(w) \cdot \pi^{2m} &= \pi^{2m} \sum_{\ell\in\mathbb{Z}} (w + \ell\beta)^2 \widehat{Q}_m(w + \ell\beta)^{2m} = \sum_{\ell\in\mathbb{Z}} (w + \ell\beta)^{2-2m} \sin\left(\pi\left(w + \ell\beta\right)\right)^{2m} \\
&= w^{2-2m} \sin(\pi w)^{2m} \\
&\quad + \sum_{\ell=1}^{\infty} \left((\ell\beta + w)^{2-2m} \sin\left(\pi\left(w + \ell\beta\right)\right)^{2m} + (\ell\beta - w)^{2-2m} \sin\left(\pi\left(\ell\beta - w\right)\right)^{2m} \right) \\
&\leq W + \sum_{\ell=1}^{\infty} \left((\ell\beta + w)^{2-2m} + (\ell\beta - w)^{2-2m} \right).
\end{aligned}$$

Using Lemma 5.2.10 as $0 < w \leq d \leq \frac{\beta}{2}$.

$$\Omega(w) \leq W + \sum_{\ell=1}^{\infty} \left((\ell\beta + d)^{2-2m} + (\ell\beta - d)^{2-2m} \right).$$

As the sum is decreasing, we can use that $\sum_{\ell=1}^{\infty} f(\ell) \leq f(1) + \int_1^{\infty} f(\ell)\, d\ell$.

$$\begin{aligned}
\Omega(w) &\leq W + (\beta + d)^{2-2m} + (\beta - d)^{2-2m} + \int_1^{\infty} (\ell\beta + d)^{2-2m} + (\ell\beta - d)^{2-2m} \, d\ell \\
&= W + (\beta + d)^{2-2m} + (\beta - d)^{2-2m} + \frac{(\beta - d)^{3-2m} + (\beta + d)^{3-2m}}{(2m-3)\,\beta} =: \Omega_1(d) \cdot \pi^{2m}.
\end{aligned}$$

As $\Omega_1(d)$ is non-decreasing for $d \in [0, \frac{\beta}{2}]$ we can state $\Omega_1\left(\frac{\beta}{2}\right)\pi^{2m}$ as a trivial upper bound of $\Omega(w)$ for $w \in [0, d]$. Hence, we can use $\Omega_1\left(\frac{\beta}{2}\right)$ with the infimum of Ξ_1 to construct an upper bound on the supremum of $B_{Q_m,1/\beta}$. Such an upper bound would be of the form

$$M_{Q_m,1/\beta} \leq \frac{\Omega_1\left(\frac{\beta}{2}\right)}{\inf \Xi_1(w)}.$$

The lower bound of the infimum of Ξ_1 is found using Lemma 5.2.8 and depends on the infimum of $\sigma_{q,m}$. Because $\inf \sigma_{q,m}$ might be many times smaller than $\sigma_{q,m}$ this method usually results in a worse bound. Further with Conjecture 5.2.7 unproven we cannot calculate $\inf \sigma_{q,m}$ for all q and m.

To avoid taking the infimum of $\sigma_{q,m}$, we need a bound for the numerator, which is also scaled by $\sigma_{q,m}$. To do so we first need to upper bound $\Omega_1(d)$.

Lemma 5.2.11 *Let $d \in]0, \frac{\beta}{2}]$. Then $\Omega_1(d) \leq \min(d^2\pi^{2m}, \pi^{2m-2}) + 2^{2m-1}\beta^{2-2m}$.*

Proof. The statement follows from $\Omega_1(d)$ being non-decreasing for $d \in [0, \frac{\beta}{2}]$ due to Lemma 5.2.10. Inserting $d = \frac{\beta}{2}$:

$$\begin{aligned}
&\Omega_1(d) \cdot \pi^{2m} &(5.6)\\
&= W + (\beta + d)^{2-2m}\left(1 + \frac{\beta + d}{\beta(2m-3)}\right) + (\beta - d)^{2-2m}\left(1 + \frac{\beta - d}{\beta(2m-3)}\right)\\
&= W + 2^{2m-2}3^{2-2m}\beta^{2-2m}\left(1 + \frac{3}{2(2m-3)}\right) + 2^{2m-2}\beta^{2-2m}\left(1 + \frac{1}{2(2m-3)}\right)\\
&= W + 2^{2m-2}\beta^{2-2m}\left(3^{2-2m}\left(1 + \frac{3}{2(2m-3)}\right) + \left(1 + \frac{1}{2(2m-3)}\right)\right)\\
&\leq \min(d^2\pi^{2m}, \pi^{2m-2}) + 2^{2m-1}\beta^{2-2m}.
\end{aligned}$$

since $3^{2-2m}\left(1 + \frac{3}{2(2m-3)}\right) + \left(1 + \frac{1}{2(2m-3)}\right) \leq 2$ for all $m \geq 2$. □

Now we consider $w \in [d, \frac{\beta}{2}]$. Then as $\beta = \frac{p}{q}$.

$$\begin{aligned}
\Omega(w) \cdot \pi^{2m} &= \sum_{\ell \in \mathbb{Z}} (w + \ell\beta)^{2-2m} \sin(\pi(w + \ell\beta))^{2m}\\
&= \sum_{\ell=1}^{\infty} ((\ell-1)\beta + w)^{2-2m} \sin(\pi(w + (\ell-1)\beta))^{2m}\\
&\quad + \sum_{\ell=1}^{\infty} (\ell\beta - w)^{2-2m} \sin(\pi(\ell\beta - w))^{2m}\\
&= \sum_{\ell=0}^{\infty}\sum_{a=1}^{q} ((\ell q + a - 1)\beta + w)^{2-2m} \sin(\pi(w + (\ell q + a - 1)\beta))^{2m}\\
&\quad + \sum_{\ell=0}^{\infty}\sum_{a=1}^{q} ((\ell q + a)\beta - w)^{2-2m} \sin(\pi((\ell q + a)\beta - w))^{2m})\\
&\leq \sum_{\ell=0}^{\infty}\sum_{a=1}^{q} (\ell q\beta + w)^{2-2m} \sin(\pi(w + (\ell q + a - 1)\beta))^{2m}\\
&\quad + \sum_{\ell=0}^{\infty}\sum_{a=1}^{q} ((\ell q + 1)\beta - w)^{2-2m} \sin(\pi((\ell q + a)\beta - w))^{2m})\\
&= \sigma_{q,m}(w) \sum_{\ell=0}^{\infty} \left((\ell q\beta + w)^{2-2m} + ((\ell q + 1)\beta - w)^{2-2m}\right)\\
&= \sigma_{q,m}(w) \sum_{\ell=0}^{\infty} \left((\ell p + w)^{2-2m} + \left(\ell p + \frac{p}{q} - w\right)^{2-2m}\right) =: (*).
\end{aligned}$$

As the sum is decreasing, we can use that $\sum_{\ell=0}^{\infty} f(\ell) \leq f(0) + \int_0^{\infty} f(\ell)\, d\ell$.

$$\begin{aligned}(*) &\leq \sigma_{q,m}(w)\Bigg(w^{2-2m} + \left(\frac{p}{q} - w\right)^{2-2m} \\ &\quad + \int_0^{\infty} (\ell p + w)^{2-2m} + \left(\ell p + \frac{p}{q} - w\right)^{2-2m} d\ell \Bigg) \\ &= \sigma_{q,m}(w) \left(w^{2-2m} + \left(\frac{p}{q} - w\right)^{2-2m} + \frac{\left(\frac{p}{q} - w\right)^{3-2m} + w^{3-2m}}{(2m-3)\,p} \right) =: \Omega_2(w) \cdot \pi^{2m}.\end{aligned}$$

Note that as $\sigma_{q,m}(w)$ is $\frac{1}{q}$-periodic, its value remains unchanged when increasing w by $\frac{1}{q}$. The second factor above is decreasing, as we can use Lemma 5.2.10 after translating w by $-\frac{p}{2q}$.

Lemma 5.2.12 *Let $w \in [d, \frac{\beta}{2}]$. Then*

$$\Omega_2(w) \cdot \pi^{2m} \leq \sigma_{q,m}(w) \left(d^{2-2m} + \left(\frac{p}{q} - d\right)^{2-2m} + \frac{\left(\frac{p}{q} - d\right)^{3-2m} + d^{3-2m}}{(2m-3)\,p} \right).$$

Proof.

$$\begin{aligned}\Omega_2(w) \cdot \pi^{2m} &\leq \sigma_{q,m}(w) \sup_{w \in]d, \frac{p}{2q}]} \left(w^{2-2m} + \left(\frac{p}{q} - w\right)^{2-2m} + \frac{\left(\frac{p}{q} - w\right)^{3-2m} + w^{3-2m}}{(2m-3)\,p} \right) \\ &= \sigma_{q,m}(w) \left(d^{2-2m} + \left(\frac{p}{q} - d\right)^{2-2m} + \frac{\left(\frac{p}{q} - d\right)^{3-2m} + d^{3-2m}}{(2m-3)\,p} \right). \qquad (5.7)\end{aligned}$$

□

We now find values of d such that Ω_2 has a supremum on $[d, \frac{p}{2q}]$ that is guaranteed to be higher than $\sup_{w \in [0,d[} \Omega(w)$. As $\Omega_1(d)$ is non-decreasing by Lemma 5.2.10, we do this by finding d and w such that $0 < d \leq w \leq \frac{\beta}{2}$ and $\Omega_1(d) \leq \Omega_2(w)$, as we then have:

$$\sup_{w \in [0,d]} \Omega(w) \leq \Omega_1(d) \leq \Omega_2(w) \leq \sup_{\omega \in \left[d, \frac{p}{2q}\right]} \Omega_2(\omega).$$

Note that the larger the value of d, the smaller the last factor of (5.7) will be, thus a large value of d is preferable.

5.2.3 Finding d

In general we will evaluate Ω_2 on some $w \in]0, \frac{p}{2q}]$. Then if $\Omega_2(w) \geq \Omega_1(d)$ for some $d \in]0, w]$, we have that when using this value of d, then $\sup_{\omega \in [d, \frac{p}{2q}]} \Omega_2(\omega) \geq \Omega_2(w) \geq \Omega_1(d)$. We will use multiple approaches to find d as to minimize the gap between $\sup_{\omega \in [d, \frac{p}{2q}]} \Omega_2(\omega)$ and $\Omega_1(d)$.

If p and q are odd, then $\frac{p}{2q} \in \arg\max \sigma_{q,m}$, otherwise exactly one of p and q is even since $\gcd(p, q) = 1$, thus $\frac{p-1}{2q} \in \arg\max \sigma_{q,m}$ by Lemma 5.2.5.

Now assume p and q are odd, then evaluating Ω_2:

$$\begin{aligned}
&\Omega_2\left(\frac{p}{2q}\right) \cdot \pi^{2m} \\
&= \sigma_{q,m}\left(\frac{p}{2q}\right)\left(\left(\frac{p}{2q}\right)^{2-2m} + \left(\frac{p}{q} - \frac{p}{2q}\right)^{2-2m} + \frac{\left(\frac{p}{q} - \frac{p}{2q}\right)^{3-2m} + \left(\frac{p}{2q}\right)^{3-2m}}{(2m-3)\,p}\right) \\
&= \sigma_{q,m}\left(\frac{p}{2q}\right) 2\left(\frac{p}{2q}\right)^{2-2m}\left(1 + \frac{\frac{p}{2q}}{(2m-3)p}\right) \\
&= \sigma_{q,m}\left(\frac{p}{2q}\right) 2^{2m-1}\left(\frac{p}{q}\right)^{2-2m}\left(1 + \frac{1}{(2m-3)2q}\right) \\
&\overset{(5.2.4)}{\geq} 2^{2m-1}\left(\frac{p}{q}\right)^{2-2m}\left(1 + \frac{1}{(2m-3)2q}\right).
\end{aligned}$$

Now assume that either p or q is even. Then evaluating Ω_2 and using Lemma 5.2.10 we get that:

$$\begin{aligned}
&\Omega_2\left(\frac{p-1}{2q}\right) \cdot \pi^{2m} \\
&= \sigma_{q,m}\left(\frac{p-1}{2q}\right)\left(\left(\frac{p-1}{2q}\right)^{2-2m} + \left(\frac{p}{q} - \frac{p-1}{2q}\right)^{2-2m} + \frac{\left(\frac{p}{q} - \frac{p-1}{2q}\right)^{3-2m} + \left(\frac{p-1}{2q}\right)^{3-2m}}{(2m-3)\,p}\right) \\
&= \sigma_{q,m}\left(\frac{p-1}{2q}\right)\left(\left(\frac{p-1}{2q}\right)^{2-2m} + \left(\frac{p+1}{2q}\right)^{2-2m} + \frac{\left(\frac{p+1}{2q}\right)^{3-2m} + \left(\frac{p-1}{2q}\right)^{3-2m}}{(2m-3)\,p}\right) \\
&= \sigma_{q,m}\left(\frac{p-1}{2q}\right) 2^{2m-2}\left(\left(\frac{p-1}{q}\right)^{2-2m} + \left(\frac{p+1}{q}\right)^{2-2m} + \frac{\left(\frac{p+1}{q}\right)^{3-2m} + \left(\frac{p-1}{q}\right)^{3-2m}}{(2m-3)\,2p}\right) \\
&\geq \sigma_{q,m}\left(\frac{p-1}{2q}\right) 2^{2m-2}\left(\left(\frac{p}{q}\right)^{2-2m} + \left(\frac{p}{q}\right)^{2-2m} + \frac{\left(\frac{p}{q}\right)^{3-2m} + \left(\frac{p}{q}\right)^{3-2m}}{(2m-3)\,2p}\right) \\
&= \sigma_{q,m}\left(\frac{p-1}{2q}\right) 2^{2m-1}\left(\frac{p}{q}\right)^{2-2m}\left(1 + \frac{\frac{p}{q}}{(2m-3)2p}\right)
\end{aligned}$$

$$\overset{(5.2.4)}{\geq} 2^{2m-1}\left(\frac{p}{q}\right)^{2-2m}\left(1+\frac{1}{(2m-3)2q}\right).$$

Note that the above is not useful for $p=1$, as $\frac{p-1}{2q}=0$ only guarantees that $\infty=\sup_{w\in[0,\frac{p}{2q}]}\Omega_2(w)\geq\Omega_1(0)$. To alleviate this we assume $\beta=\frac{p}{q}>1$.

Thus, with the upper bound from Lemma 5.2.11 we can easily find some $d>0$ such that either $\Omega_2\left(\frac{p}{2q}\right)\geq\Omega_1(d)$ or $\Omega_2\left(\frac{p-1}{2q}\right)\geq\Omega_1(d)$. Eliminating the need for $\Omega_1(d)$.

Using the proof of Lemma 5.2.11 we then get the following:

$$\begin{aligned}
&\Omega(w)\cdot\pi^{2m}\leq\Omega_1(d)\cdot\pi^{2m}\\
&=d^2\pi^{2m}+2^{2m-2}\beta^{2m-2}\left(3^{2-2m}\left(1+\frac{3}{2(2m-3)}\right)+\left(1+\frac{1}{2(2m-3)}\right)\right)\\
&\leq\sup\sigma_{q,m}\cdot 2^{2m-1}\beta^{2-2m}\left(1+\frac{1}{(2m-3)2q}\right)\leq\sup_{w\in\left[d,\frac{p}{2q}\right]}\Omega_2(w)\cdot\pi^{2m}.
\end{aligned}$$

Solving for d in the second inequality in the above, we get a value of d ensuring Ω_2 on $[d,\frac{p}{2q}]$ is an upper bound for $\sup_{w\in[0,]}\Omega(w)$

$$\begin{aligned}
d=\min\Bigg(&\frac{p}{2q},\ \pi^{-m}\beta^{1-m}2^{m-1}\\
&\cdot\sqrt{\sup\sigma_{q,m}\cdot\left(2+\frac{2}{(2m-3)2q}\right)-3^{2-2m}\left(1+\frac{3}{2(2m-3)}\right)-\left(1+\frac{1}{2(2m-3)}\right)}\Bigg).
\end{aligned}\tag{5.8}$$

Thus, for a fixed q we can easily find a $d>0$ for any p where $\gcd(p,q)=1$. As $\Omega_2(w)\leq\Omega_2\left(w-\frac{1}{q}\right)$ we can get even larger values of d by evaluating Ω_2 at smaller values of $w-\frac{n}{q}$ where $n\in\mathbb{Z}_{\geq0}$, with the trade-off that $d\leq w-\frac{n}{q}$.

5.2.4 Final result

We can now write an upper bound of $B_{Q_m,1/\beta}$ that does not depend on $\sigma_{q,m}$:

$$\begin{aligned}
B_{Q_m,1/\beta}(w)\leq B^*_{Q_m,1/\beta}(w)&:=\frac{\sigma_{q,m}(w)\left(w^{2-2m}+\left(\frac{p}{q}-w\right)^{2-2m}+\frac{\left(\frac{p}{q}-w\right)^{3-2m}+w^{3-2m}}{(2m-3)p}\right)}{\sigma_{q,m}(w)\frac{\left(p+w-\frac{p}{q}\right)^{1-2m}+(p-w)^{1-2m}}{(2m-1)p}}\\
&=\frac{\left(w^{2-2m}+\left(\frac{p}{q}-w\right)^{2-2m}+\frac{\left(\frac{p}{q}-w\right)^{3-2m}+w^{3-2m}}{(2m-3)p}\right)(2m-1)\,p}{\left(p+w-\frac{p}{q}\right)^{1-2m}+(p-w)^{1-2m}}.
\end{aligned}$$

To find $M^*_{Q_m,1/\beta}:=\sup_{w\in[d,\frac{p}{2q}]}B^*_{Q_m,1/\beta}(w)$ we state the following conjecture:

Conjecture 5.2.13 *$B^*_{Q_m,\frac{1}{\beta}}(w)$ is convex as a function of w on $[d, \frac{p}{2q}]$, and thus that it takes its supremum on $[d, \frac{p}{2q}]$ at either d or $\frac{p}{2q}$.*

This allows us to find $M^*_{Q_m,1/\beta}$ as $\max\{B^*_{Q_m,1/\beta}(d), B^*_{Q_m,1/\beta}\left(\frac{p}{2q}\right)\}$. If this conjecture does not hold we can replace the numerator and denominator with their supremum and infimum respectively, i.e. Lemma 5.2.12 and Lemma 5.2.8 with $\sigma_{q,m}$ cancelled out, giving us a bound that does not depend on conjectures.

$$B^*_{Q_m,\frac{1}{\beta}}(w) \leq \pi^{2m}(2m-1)\frac{d^{2-2m} + \left(\frac{p}{q} - d\right)^{2-2m} + \frac{\left(\frac{p}{q}-d\right)^{3-2m}+d^{3-2m}}{(2m-3)p}}{p^{-2m}q^{2m}\left(2q-1\right)^{1-2m}}. \tag{5.9}$$

With d as given in Eq. (5.8).

To show the difference between some of the bounds found above, we present the following plot of the frame set of Q_m. Note that the unproven parts of Conjecture 5.2.13 and Conjecture 5.2.7 are not assumed. The bounds in Eq. (5.5) and Theorem 5.2.9 are decaying too quickly for a comparison between the plotted bounds to be meaningful and are thus not included in the plot below.

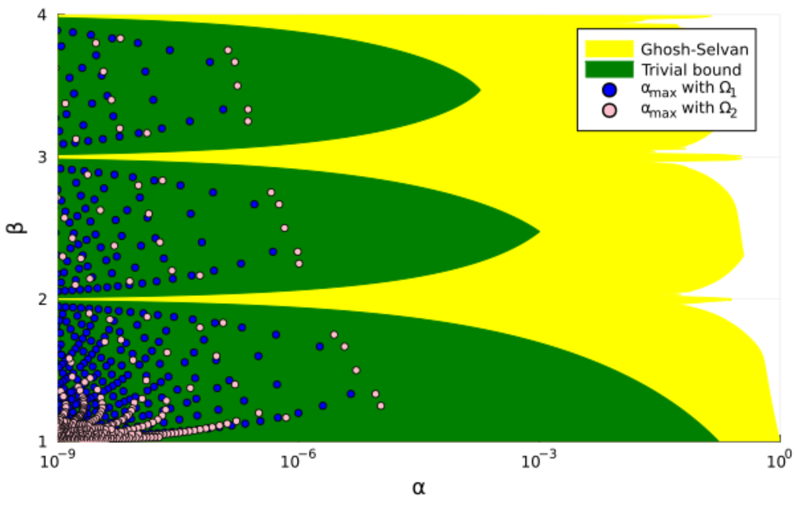

Figure 5.1: The computed frame set of Q_4 with logaritmic x-axis. The yellow area is the frame set found using the numeric implementation of the Ghosh-Selvan method as presented in Section 6.2, the green area is the frame set found by using the non-closed form in Eq. (5.2). The blue and pink dots represent the lower bounds on $\alpha_{\max}$ using $M_{Q_m,1/\beta} < \frac{\Omega_1\left(\frac{p}{2q}\right)}{\inf \Xi_1}$ and $M_{Q_m,1/\beta} < \frac{\sup_{w\in[d,\frac{p}{2q}]}\Omega_2(w)}{\inf \Xi_1}$ respectively for values of $\beta \in [1, 4]$ where $q \in \{1, ..., 100\}$. The values of d are found using a combination of the method introduced above and binary search. The plot is generated in **scripts/gs_guaranteed_bound.jl**.

5.3 Future directions

If we had more time we would look into the following ways of improving the above bounds.

5.3.1 Make a small adjustment in the way we derive Ω_2

When deriving Ω_2 we keep every term inside the sum, if instead we take the $\ell = 0$ term and upper bound it with

$$w^{2-2m}\sin(\pi w)^{2m} \leq \sigma_{q,m}(w)w^{2-2m}$$

as $\sin(\pi w)^{2m}$ is a term in $\sigma_{q,m}$. This would allow us to find the slightly smaller bound,

$$\begin{aligned}\tilde{\Omega}_2(w) = \sigma_{q,m}(w)\Bigg(& w^{2-2m} + (p+w)^{2-2m}\left(1 + \frac{p+w}{p\,(-3+2m)}\right) \\ & + (p-w)^{2-2m}\left(1 + \frac{p-w}{p\,(-3+2m)}\right)\Bigg).\end{aligned}$$

Trying it out numerically we find that $\frac{\Omega_2(w)}{\tilde{\Omega}_2(w)}$ is an increasing function going from 1 at $w = 0$ to 2 at $w = \frac{\beta}{2}$.

This variant was abandoned in favor of the slightly simpler Ω_2.

5.3.2 Using the derivative of $B_{Q_m,1/\beta}(w)$

Potentially, a better value of d could be found by letting d be the smallest $w > 0$ where $B'_{Q_m,1/\beta}(w) \geq 0$ no longer holds. An approach to this would be to define $N_\beta(w)$ and $D_\beta(w)$ such that $B_{Q_m,1/\beta} = \frac{N_\beta(w)}{D_\beta(w)}$. Then as $B'_{Q_m,h}(w) \propto N'(w)D(w) - N(w)D'(w)$ the problem can be reduced to finding when $N'(w)D(w) - N(w)D'(w)$ stops being non-negative.

5.3.3 Eliminating the need for $M_{Q_m,1/\beta}$

Another thing is to not introduce $M_{\phi,h}$ at all during the proof of Theorem 4.1.2, but instead keep $B_{\phi,h}$ and use the Cauchy–Schwarz inequality (i.e. Hölder's inequality for $p = 2$):

$$\begin{aligned}\|f'\|^2 &\leq 4\pi^2 \int_{-\infty}^{\infty} B_{\phi,h}(\omega)\left|\sum_{k\in\mathbb{Z}} d_k e^{-2\pi i h k \omega}\widehat{\phi}(\omega)\right|^2 \, d\omega \\ &\leq 4\pi^2 \left\| B_{\phi,h}\sum_{k\in\mathbb{Z}} d_k E_{-hk}\widehat{\phi}\right\| \|f\|,\end{aligned}$$

and then bound $\|B_{\phi,h}\widehat{f}\|$ as a multiple of $\|f\|$, either for arbitrary $\widehat{\phi}$ or for the B-spline specific $\widehat{Q}_m$.

Chapter 6

Numerical analysis

In this chapter, we explain our implementation of the Ghosh-Selvan and Zibulski-Zeevi methods for computing frame bounds and give a comparison of these bounds, as well as some pros and cons of each method. The Ghosh-Selvan method is a lot faster to compute, but can only tell if $(\alpha, \beta) \in \mathfrak{F}(\phi)$ but not $(\alpha, \beta) \notin \mathfrak{F}(\phi)$, while the Zibulski-Zeevi method can tell both, but is limited to the cases were $\alpha\beta = \frac{p}{q}$ for some $p, q \in \mathbb{Z}_{>0}$.

Our code implementation was originally written in Python, using the library Numpy for numerical computations, but we changed the language to Julia. The Julia implementation runs faster since it is Just-In-Time compiled. The Julia implementation also has support for arbitrary precision floating-point numbers and typed function parameters to ensure correctness. Both the Python and Julia versions save the computed values as Numpy `.npz`-files[1], thus we can do the computation in Julia and plot the data in either language. A link to a git repository containing the code used in this thesis can be found in the appendix at the start of Section 9.1.

6.1 Julia code

We will go through the code for the Ghosh-Selvan and the Zibulski-Zeevi method for computing frame bounds. The code for the Lyubarskii-Nes method is similar to the code for the Zibulski-Zeevi method, so we will therefore not go through the code for that method for the sake of conciseness, but the code is available in the appendix.

We use the following libraries imported in the file **utils.jl** throughout the code:

```julia
using LinearAlgebra
using Plots
using DoubleFloats
using Optim
using NP
import CSV
```

The libraries `LinearAlgebra` and `Plots` are Julia's default libraries for linear algebra operations and plotting, respectively. The library `DoubleFloats` allows us to work with floating-point numbers of type `Double64`, which is two `Float64` combined giving

[1]https://numpy.org/doc/stable/reference/generated/numpy.savez.html

us more precision than `Float64`, but still with the speed of `Float64` operations. The library `Optim` is an optimization library for optimizing functions and the `NPZ` and `CSV` libraries allow us to save and load files in the Numpy NPZ-file format and as comma separated values, respectively.

We start by explaining some basic functions from the file **utils.jl** that is used in the code for the other methods.

The code for the Ghosh-Selvan and Zibulski-Zeevi methods can give values of A and B that does not satisfy $0 < A \leq B < \infty$ and is therefore not frame bounds. We have implemented a small function `is_gabor_frame` that checks if A and B are frame bounds within some tolerance.

```julia
function is_gabor_frame(A::Real, B::Real; min_tol::Real = 0, max_tol::Real =
    Inf)::Bool
    return min_tol < A <= B < max_tol
end
```

The arguments `min_tol` and `max_tol` are optional keyword arguments, which can be used to specify the minimum and maximum tolerance for `min_tol` $< A \leq B <$ `max_tol`. Since our computations are done with floating-point numbers, we will not get exact results especially when our values are very close to zero or very large, and therefore a minimum and maximum tolerance are needed.

The function `uniform_spaced_values` takes the parameters $x_1 \in \mathbb{R}$, $x_n \in \mathbb{R}$, $d_x \in \mathbb{R}$ where $x_n > x_1$ and $d_x > 0$. It also takes a type `T` that determines what type of floating point number, i.e. the precision, we want our values to have. The function returns the shortest vector of real numbers $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ such that $x_{i+1} - x_i = c \leq d_x$ for all i and some positive constant c.

```julia
function uniform_spaced_values(
    x1::Real,
    xn::Real,
    dx::Real;
    T::Type{<:Real} = Float64
)::Vector{T}
    @assert dx > 0 && x1 < xn

    n = ceil(Int, (xn - x1) / dx) + 1
    return collect(T, range(x1, stop=xn, length=n))
end
```

The function `bspline` takes $m \in \mathbb{N}$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ as input and computes $y_i = Q_m(x_i)$ for all i using Proposition 2.3.3, returning $\boldsymbol{y} = (y_1, \ldots, y_n)$.

```julia
function bspline(m::Integer, x::Vector{T})::Vector{T} where T<:Real
    @assert m > 0

    y = typeof(x)((-m/2 .<= x) .& (x .<= m/2))
    if m >= 2
        y .*= sum((-1)^j * binomial(m, j) * max.(x .+ (m/2 - j), 0).^(m - 1) for
            j in 0:m) / factorial(m - 1)
    end

    @assert size(x) == size(y)
    return y
end
```

Observe that in the code, we use the "."-syntax called broadcasting in Julia. This can be used to multiply two vectors together element-wise or add a constant to all elements in a vector. The "."-syntax can be used on any function or operator in Julia to do element-wise operations on one or more arrays.

We use the `Optim` library for minimizing and maximizing functions. We have defined two wrapper-functions that call `Optim.optimize` and `Optim.minimum` with our desired parameters. The function `minimize_bounded` takes

- $f : \mathbb{R}^n \to \mathbb{R}$ the function that shall be minimized,
- $\boldsymbol{a} = (a_1, \dots, a_n) \in \mathbb{R}^n$ and $\boldsymbol{b} = (b_1, \dots, b_n) \in \mathbb{R}^n$ which specifies the finite interval $[a_1, b_1] \times \cdots \times [a_n, b_n]$ on which f should be minimized,
- $\boldsymbol{x}_0 \in [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$ the initial point,
- $y_0 \in \mathbb{R}$ such that $y_0 = f(\boldsymbol{x}_0)$,
- `optim_iter` is the number of iterations the optimization algorithm should run,
- `method` is the optimization algorithm used from the library `Optim`,
- `print_error` specifies if errors from `Optim` should be printed.

It tries to minimize f and returns $y = f(\boldsymbol{x})$ for some $\boldsymbol{x} \in [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$ such that $y \leq y_0$.

```julia
function minimize_bounded(
    f::Function,
    a::Vector{T},
    b::Vector{T},
    x0::Vector{T},
    y0::T,
    optim_iter::Integer;
    method::M = LBFGS(),
    print_error::Bool = false,
)::T where {T<:Real, M<:Optim.AbstractOptimizer}
    @assert a < b && optim_iter > 0

    try
        y = minimum(optimize(f, prevfloat.(a), nextfloat.(b), x0, Fminbox(method),
            Optim.Options(iterations = optim_iter, outer_iterations = optim_iter)
        ))
        return min(y, y0)
    catch e
        if print_error
            println("Optim error: ", e)
        end
        return y0
    end
end
```

We use `prevfloat.(a)` and `nextfloat.(b)`, since the bounded optimizer in `Optim` assumes that f should be minimized on $]a_1, b_1[\times \cdots \times]a_n, b_n[$, which does not include the end points a_i and b_i.

The function `maximize_bounded` is the negation of `minimize_bounded` and takes the same parameters. It tries to maximize f and returns $y = f(\boldsymbol{x})$ for some $\boldsymbol{x} \in [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$ such that $y \geq y_0$.

```julia
function maximize_bounded(
    f::Function,
    a::Vector{T},
    b::Vector{T},
    x0::Vector{T},
    y0::T,
    optim_iter::Integer;
    method::M = LBFGS(),
    print_error::Bool = true,
)::T where {T<:Real, M<:Optim.AbstractOptimizer}
    return -minimize_bounded(x -> -f(x), a, b, x0, -y0, optim_iter; method,
        print_error)
end
```

We have defined very similar functions for unbounded minimization and maximization called `minimize_unbounded` and `maximize_unbounded` in the file **utils.jl**, which we skip here for the sake of conciseness. They take the same parameters except the unbounded versions do not take $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^n$.

6.2 Ghosh-Selvan method in Julia

We are now ready to look at the Ghosh-Selvan implementation in the file **modules/ghosh_selvan.jl**. We start by computing $B_{\phi,1/\beta}(w)$ from Definition 4.1.1 and the B-spline special case $B_{Q_m,1/\beta}(w)$ from Theorem 4.2.4. We assume that the $\widehat{\phi}$ function has compact support or sufficient decay such that the sums in $B_{\phi,1/\beta}(w)$ can be evaluated as a sum with a finite number of terms. We can still get good results with this method on functions for which the Fourier transform does not have compact support if they decay so rapidly that the values are sufficiently close to zero and decrease quickly outside the defined compact support. An example of this is the B-spline of order $m \geq 2$ where $|\widehat{Q}_m(w)| = |\frac{\sin(w\pi)}{w\pi}|^m \leq (\pi w)^{-m}$. Note that for every $\phi \in \mathcal{A}$ then (ii) in Definition 4.1.1 ensures that ϕ decays.

The function `compute_sums_generic` takes $\widehat{\phi}$ where $\phi \in \mathcal{A}$, its compact support $(a, b) \in \mathbb{R}^2$ where $\operatorname{supp} \widehat{\phi} = [a, b]$, $\beta \in \mathbb{R}$, and $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ as input and computes

$$s_1(w_i) = \sum_{k=\lfloor \frac{a-w}{\beta} \rfloor}^{\lceil \frac{b-w}{\beta} \rceil} (w_i + k\beta)^2 |\widehat{\phi}(w_i + k\beta)|^2 \quad \text{and} \quad s_2(w_i) = \sum_{k=\lfloor \frac{a-w}{\beta} \rfloor}^{\lceil \frac{b-w}{\beta} \rceil} |\widehat{\phi}(w_i + k\beta)|^2$$

assuming that $\operatorname{supp} \widehat{\phi} = [a, b]$, as $a \leq w_i + k\beta \leq b$ then implies $\frac{a-w_i}{\beta} \leq k \leq \frac{b-w_i}{\beta}$. Note that $B_{\phi,1/\beta}(w_i) = \frac{s_1(w_i)}{s_2(w_i)}$. The function returns $\boldsymbol{s}_1 = (s_1(w_1), \ldots, s_1(w_n))$ and $\boldsymbol{s}_2 = (s_2(w_1), \ldots, s_2(w_n))$.

```julia
function compute_sums_generic(
    phi_hat::Function,
    compact_support::Tuple{Real, Real},
    beta::Real,
    w::Vector{T}
)::Tuple{Vector{T}, Vector{T}} where T<:Real
    a, b = compact_support
    @assert beta > 0 && a < b

    kmin = floor(Int, (a - maximum(w)) / beta)
    kmax = ceil(Int, (b - minimum(w)) / beta)

    s1 = sum((w .+ k*beta).^2 .* abs2.(phi_hat(w .+ k*beta)) for k in kmin:kmax)
    s2 = sum(abs2.(phi_hat(w .+ k*beta)) for k in kmin:kmax)

    @assert size(w) == size(s1) == size(s2)
    return s1, s2
end
```

We split up the numerator and denominator of the fraction $B_{\phi,1/\beta}(w) = \frac{s_1(w)}{s_2(w)}$, as we need $M_{\phi,1/\beta} = \operatorname{ess\,sup}_{w\in[0,\beta]} B_{\phi,1/\beta}(w)$, $\operatorname{ess\,inf}_{w\in[0,\beta]} s_2(w)$, and $\operatorname{ess\,sup}_{w\in[0,\beta]} s_2(w)$ to compute the frame bounds A and B.

For the B-spline special case from Theorem 4.2.4, we use the function `compute_sums_bspline` which takes $m \in \mathbb{Z}_{>0}$, $\beta \in \mathbb{R}_{>0}$, and $\boldsymbol{w} = (w_1, \dots, w_n)$ and computes

$$s_1(w_i) = \frac{1}{2\pi^2\beta}\Bigg(Q_{2m-2}(0) - Q_{2m-2}(1) + \sum_{n=1}^{\lceil m\beta\rceil}\left(2Q_{2m-2}\left(\frac{n}{\beta}\right) - Q_{2m-2}\left(\frac{n}{\beta}+1\right) - Q_{2m-2}\left(\frac{n}{\beta}-1\right)\right)\cos(2\pi n w_i/\beta)\Bigg)$$

and

$$s_2(w_i) = \frac{1}{\beta}\left(Q_{2m}(0) + 2\sum_{n=1}^{\lceil m\beta\rceil} Q_{2m}\left(\frac{n}{\beta}\right)\cos(2\pi n w_i/\beta)\right).$$

Note that $B_{\phi,1/\beta}(w_i) = \frac{s_1(w_i)}{s_2(w_i)}$. The function returns $\boldsymbol{s}_1 = (s_1(w_1), \dots, s_1(w_n))$, and $\boldsymbol{s}_2 = (s_2(w_1), \dots, s_2(w_n))$.

```julia
function compute_sums_bspline(
    m::Integer,
    beta::Real,
    w::Vector{T}
)::Tuple{Vector{T}, Vector{T}} where T<:Real
    @assert m > 0 && beta > 0

    n = collect(T, 1:ceil(Int, m * beta))
    x = n ./ beta
    cos_vals = cos.(((2 * pi / beta) .* w) * n')

    s1 = vec(((bspline(2*m-2, T(0)) - bspline(2*m-2, T(1))) .+ sum(cos_vals .*
        (2*bspline(2*m-2, x)
       - bspline(2*m-2, x .+ 1) - bspline(2*m-2, x .- 1))', dims=2)) / (2 * pi^2
           * beta))
    s2 = vec((bspline(2*m, T(0)) .+ 2*sum(cos_vals .* bspline(2*m, x)', dims=2))
        / beta)

    @assert size(w) == size(s1) == size(s2)
    return s1, s2
end
```

Note that $\boldsymbol{x}'$ means transposing a vector $\boldsymbol{x}$ in Julia, thus in `cos.(((2 * pi / beta) .* w) * n')` we first compute the outer product `((2 * pi / beta) .* w) * n'` and then compute the cosine of each element in the matrix created by the outer product.

Now that we can compute $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, we can approximate $M_{\phi,1/\beta}$ with the following Julia function. The function `compute_M` takes:

- `compute_sums` which is either `compute_sums_generic` or `compute_sums_bspline` defined above,
- $\beta \in \mathbb{R}$,
- $d_w \in \mathbb{R}$ which specifies $w_{i+1} - w_i \leq d_w$ for all i,
- `B_symmetric` which specifies if $B_{\phi,1/\beta}(w)$ is symmetric over the interval $[0, \beta]$,
- `ensure_integer_w` which adds the integers from 1 to $\lfloor \beta \rfloor$ to the vector $\boldsymbol{w}$,
- `tol_M` is the allowed tolerance when ensuring that $2\sqrt{M} > \beta$,
- `optim_iter` which is the number of iterations the optimizer algorithm runs,
- `T2` which is the type of floating-point number used for calculations.

The function approximates $M \approx M_{\phi,1/\beta} = \operatorname{ess\,sup}_{w\in[0,\beta]} B_{\phi,1/\beta}(w)$ and returns M, $\boldsymbol{w}$, and $\boldsymbol{s}_2$. Here $\boldsymbol{w}$ and $\boldsymbol{s}_2$ are needed for computing M and are returned for later use.

```julia
function compute_M(
    compute_sums::Function,
    beta::Real,
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{T2, Vector{T2}, Vector{T2}}
    @assert optim_iter_M >= 0

    b = B_symmetric ? beta / 2 : beta
    w = uniform_spaced_values(0, b, dw; T=T2)

    if ensure_integer_w && beta > 1
        w = vcat(w, 1:floor(beta))
    end

    s1, s2 = compute_sums(beta, w)
    (M, idx_M) = findmax(s1 ./ s2)

    if optim_iter_M > 0
        f(w) = compute_sums(beta, w) |> x -> x[1][1] / x[2][1]
        M = maximize_bounded(f, [T2(0)], [T2(b)], [w[idx_M]], M, optim_iter_M)
    end

    if beta^2 >= 4*M + tol_M
        M = NaN
    end

    @assert size(w) == size(s2)
    return M, w, s2
end
```

The variable `B_symmetric` specifies if B is symmetric on the interval $[0, \beta]$, as we then only need to find the maximum on $[0, \frac{\beta}{2}]$, which saves us half the computations. We compute $\boldsymbol{w} = (w_1, \ldots, w_n)$, $B_{\phi,1/\beta}(w_i) = \frac{s_1(w_i)}{s_2(w_i)}$, and $M = \max_{w_i} B_{\phi,1/\beta}(w_i)$ for each i and then we try to maximize $M \approx \max_{w \in [0,b]} B_{\phi,1/\beta}(w)$ with the library `Optim` if `opim_iter` > 0. We also check that the computed M satisfies $2\sqrt{M} < \beta$ with some allowed tolerance, since we know that $\alpha\beta < 1$ and $\alpha < \frac{1}{2\sqrt{M}}$ from Theorem 4.1.9.

The function `compute_min_max_s2` is very similar to `compute_M` and takes:

- `compute_sums` which is either `compute_sums_generic` or `compute_sums_bspline`,
- $\beta \in \mathbb{R}$,
- $\boldsymbol{w} = (w_1, \dots, w_n) \in \mathbb{R}^n$,
- $\boldsymbol{s}_2 = (s_{2,1}, \dots, s_{2,n}) \in \mathbb{R}^n$,
- `optim_iter` which is the number of iterations the optimizer algorithm runs.

It returns $s_{2,\min} \leq \min_i s_{2,i}$ and $s_{2,\max} \geq \max_i s_{2,i}$.

```julia
function compute_min_max_s2(
    compute_sums::Function,
    beta::Real,
    w::Vector{T2},
    s2::Vector{T2};
    optim_iter_s2::Integer = 0
)::Tuple{T2, T2} where T2<:Real
    @assert optim_iter_s2 >= 0 && size(w) == size(s2)

    (s2_min, idx_s2_min) = findmin(s2)
    (s2_max, idx_s2_max) = findmax(s2)

    if optim_iter_s2 > 0
        f(w) = compute_sums(beta, w)[2][1]
        b = w[end]

        s2_min = minimize_bounded(f, [T2(0)], [T2(b)], [w[idx_s2_min]], s2_min,
            optim_iter_s2)
        s2_max = maximize_bounded(f, [T2(0)], [T2(b)], [w[idx_s2_max]], s2_max,
            optim_iter_s2)
    end

    return s2_min, s2_max
end
```

Now we have the tools to compute the frame set and frame bounds for a fixed β. We know from Theorem 4.1.9 that if $0 < \alpha < \frac{1}{2\sqrt{M_{\phi,1/\beta}}}$ then $\mathcal{G}(\phi, \alpha, \beta)$ forms a frame. Thus, we can calculate $\frac{1}{2\sqrt{M_{\phi,1/\beta_i}}}$ for different β_i to find the corresponding maximum α_i.

The function `frame_set_max_alpha` takes:

- `compute_sums` which is either `compute_sums_generic` or `compute_sums_bspline`,
- $\boldsymbol{\beta} = (\beta_1, \dots, \beta_n) \in \mathbb{R}^n$,
- $d_w \in \mathbb{R}$ which specifies $w_{i+1} - w_i \leq d_w$ for all i,
- `B_symmetric` which specifies if $B_{\phi,1/\beta}(w)$ is symmetric over the interval $[0, \beta]$,
- `ensure_integer_w` which adds the integers from 1 to $\lfloor \beta \rfloor$ to the vector $\boldsymbol{w}$,
- `tol_M` is the allowed tolerance when ensuring that $2\sqrt{M} < \beta$,
- `optim_iter_M` which is the number of iterations the optimizer algorithm runs,
- `print_progess` which specifies if progress should be printed,
- `parallelize` which specifies if the code should run parallelized
- `T2` which is the type of floating point number used for calculations.

It returns $\boldsymbol{\alpha}_{\max} = (\alpha_{\max,1}, \dots, \alpha_{\max,n})$ where $\alpha_{\max,i} \approx \frac{1}{2\sqrt{M_{\phi,1/\beta_i}}}$.

```julia
function frame_set_max_alpha(
    compute_sums::Function,
    beta::Vector{T1},
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    print_progress::Integer = 0,
    parallelize::Bool = false,
    T2::Type{<:Real} = Float64,
)::Vector{T1} where T1<:Real
    @assert print_progress >= 0

    alpha_max = zeros(length(beta))

    I = length(beta)
    function f(i, beta_i)
        if print_progress > 0
            println("Iteration beta = $(beta_i) ($i out of $I)")
        end

        M, _, _ = compute_M(compute_sums, beta_i, dw; B_symmetric,
            ensure_integer_w, tol_M, optim_iter_M, T2)

        if isfinite(M) && M > 0
            alpha_max[i] = 1/(2*sqrt(M))
        end
    end

    if parallelize
        Threads.@threads for i in axes(beta, 1)
            f(i, beta[i])
        end
    else
        for (i, beta_i) in enumerate(beta)
            f(i, beta_i)
        end
    end

    @assert size(alpha_max) == size(beta)
    return alpha_max
end
```

The above implementation has many calculations, where element i in some input vector is used to calculate element i in the output, without using or changing any other indices. This makes these computations great for vectorization, Parallel Computing (PC), and High-Performance computing (HPC). In the above function, we use the Julia macro `Threads.@threads` to parallelize the for-loop if the `parallelize` variable is set to `true`. We have applied HPC and PC in the scripts files **scripts/peaks.jl** and **scripts/gs_guaranteed_bound.jl** respectively, to significantly speed up computations of the Ghosh-Selvan method when running on multiple CPU cores.

We can use Theorem 4.1.9 to approximate the frame bounds

$$A = \frac{1}{\alpha}(1 - 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \operatorname*{ess\,inf}_{\xi\in[0,\beta]} \sum_{k\in\mathbb{Z}} |\widehat{\phi}(\xi+\beta k)|^2 \approx \frac{1}{\alpha}(1-2\alpha\sqrt{M})^2 \min_i s_2(w_i)$$

$$B = \frac{1}{\alpha}(1 + 2\alpha\sqrt{M_{\phi,1/\beta}})^2 \operatorname*{ess\,sup}_{\xi\in[0,\beta]} \sum_{k\in\mathbb{Z}} |\widehat{\phi}(\xi+\beta k)|^2 \approx \frac{1}{\alpha}(1+2\alpha\sqrt{M})^2 \max_i s_2(w_i).$$

Note that these estimated values of the frame bounds are larger and smaller than the theoretical ones in Theorem 4.1.9, respectively.

The function `frame_bounds_fixed_beta` takes:

- `compute_sums` which is either `compute_sums_generic` or `compute_sums_bspline`,
- $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{R}^n$,
- $\beta \in \mathbb{R}$,
- $d_w \in \mathbb{R}$ which specifies $w_{i+1} - w_i \le d_w$ for all i,
- `B_symmetric` which specifies if $B_{\phi,1/\beta}(w)$ is symmetric over the interval $[0, \beta]$,
- `ensure_integer_w` which adds the integers from 1 to $\lfloor\beta\rfloor$ to the vector $\boldsymbol{w}$,
- `tol_M` is the allowed tolerance when ensuring that $2\sqrt{M} < \beta$,
- `optim_iter_M` which is the number of iterations run by the optimizer to estimate $M_{\phi,1/\beta}$,
- `optim_iter_s2` which is the number of iterations run by the optimizer to estimate the minimum and maximum of s_2,
- `T2` which is the type of floating point number used for calculations.

It returns $\boldsymbol{A} = (A_1, \dots, A_k)$, $\boldsymbol{B} = (B_1, \dots, B_k)$, and $\boldsymbol{\alpha}' = (\alpha'_1, \dots, \alpha'_k)$ where $k \le n$ and A_i and B_i are frame bounds for $\mathcal{G}(\phi, \alpha'_i, \beta)$ for all i and a fixed β.

```julia
function frame_bounds_fixed_beta(
    compute_sums::Function,
    alpha::Vector{T1},
    beta::Real,
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    optim_iter_s2::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}} where T1<:Real
    M, w, s2 = compute_M(compute_sums, beta, dw; B_symmetric, ensure_integer_w,
        tol_M, optim_iter_M, T2)

    if isfinite(M) && M > 0
        alpha = alpha[alpha .< 1/(2*sqrt(M))]
        s2_min, s2_max = compute_min_max_s2(compute_sums, beta, w, s2;
            optim_iter_s2)

        c = 2 * sqrt(M)
        A = Vector{T1}((1 .- c .* alpha).^2 .* s2_min ./ alpha)
        B = Vector{T1}((1 .+ c .* alpha).^2 .* s2_max ./ alpha)
    else
        A, B, alpha = T1[], T1[], T1[]
    end

    @assert size(A) == size(B) == size(alpha)
    return A, B, alpha
end
```

The next function runs `frame_bounds_fixed_beta` on multiple β values. The function `frame_bounds` takes:

- `compute_sums` which is either `compute_sums_generic` or `compute_sums_bspline`,
- $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{R}^n$,
- $\boldsymbol{\beta} = (\beta_1, \dots, \beta_m) \in \mathbb{R}^m$,
- $d_w \in \mathbb{R}$ which specifies $w_{i+1} - w_i \leq d_w$ for all i.
- `B_symmetric` which specifies if $B_{\phi,1/\beta}(w)$ is symmetric over the interval $[0, \beta]$,
- `ensure_integer_w` which adds the integers from 1 to $\lfloor \beta \rfloor$ to the vector $\boldsymbol{w}$,
- `tol_M` is the allowed tolerance when ensuring that $2\sqrt{M} < \beta$,
- `optim_iter_M` which is the number of iterations run by the optimizer to estimate $M_{\phi,1/\beta}$,
- `optim_iter_s2` which is the number of iterations run by the optimizer to estimate the minimum and maximum of s_2,
- `print_progess` which specifies if progress should be printed,
- `T2` which is the type of floating point number used for calculations.

It returns $\boldsymbol{A} = (A_1, \dots, A_k)$, $\boldsymbol{B} = (B_1, \dots, B_k)$, $\boldsymbol{\alpha}' = (\alpha_1', \dots, \alpha_k')$, and $\boldsymbol{\beta}' = (\beta_1', \dots, \beta_k')$ where A_i and B_i are the frame bounds for $\mathcal{G}(\phi, \alpha_i', \beta_i')$ for all i where $\mathcal{G}(\phi, \alpha_i', \beta_i')$ is a frame.

```julia
function frame_bounds(
    compute_sums::Function,
    alpha::Vector{T1},
    beta::Vector{T1},
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    optim_iter_s2::Integer = 0,
    print_progress::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}, Vector{T1}} where T1<:Real
    A = T1[]
    B = T1[]
    alpha_new = T1[]
    beta_new = T1[]

    I = length(beta)
    for (i, beta_i) in enumerate(beta)
        if print_progress > 0
            println("Iteration beta = $(beta_i) ($i out of $I)")
        end

        Ai, Bi, alpha_i = frame_bounds_fixed_beta(compute_sums, alpha, beta_i,
            dw; B_symmetric, ensure_integer_w, tol_M, optim_iter_M,
            optim_iter_s2, T2)
        N = length(Ai)

        if N > 0
            append!(A, Ai)
```

```
                append!(B, Bi)
                append!(alpha_new, alpha_i)
                append!(beta_new, fill(beta_i, N))
            end
        end

        @assert size(A) == size(B) == size(alpha_new) == size(beta_new)
        return A, B, alpha_new, beta_new
end
```

The next function simply runs `frame_bounds` on a regular grid of α and β values. The function `frame_bounds_grid` takes:

- `compute_sums` which is either `compute_sums_generic` or `compute_sums_bspline`,
- d_α which specifies $\alpha_{i+1} - \alpha_i \leq d_\alpha$ for all i,
- d_β which specifies $\beta_{i+1} - \beta_i \leq d_\beta$ for all i,
- $\alpha_{\text{max}} \in \mathbb{R}$ specifies α_n,
- $\beta_{\text{max}} \in \mathbb{R}$ specifies β_m,
- $\alpha_{\text{min}} \in \mathbb{R}$ specifies α_1,
- $\beta_{\text{min}} \in \mathbb{R}$ specifies β_1,
- `B_symmetric` which specifies if $B_{\phi,1/\beta}(w)$ is symmetric over the interval $[0, \beta]$,
- `ensure_integer_w` which adds the integers from 1 to $\lfloor\beta\rfloor$ to the vector $\boldsymbol{w}$,
- `tol_M` is the allowed tolerance when ensuring that $2\sqrt{M} < \beta$,
- `optim_iter_M` which is the number of iterations run by the optimizer to estimate $M_{\phi,1/\beta}$,
- `optim_iter_s2` which is the number of iterations run by the optimizer to estimate the minimum and maximum of s_2,
- `print_progess` which specifies if progress should be printed,
- `T1` which is the type of floating point number that is returned,
- `T2` which is the type of floating point number used for calculations.

It returns $\boldsymbol{A} = (A_1, \ldots, A_k), \boldsymbol{B} = (B_1, \ldots, B_k)$, $\boldsymbol{\alpha}' = (\alpha_1', \ldots, \alpha_k')$, and $\boldsymbol{\beta}' = (\beta_1', \ldots, \beta_k')$ where A_i and B_i are the frame bounds for $\mathcal{G}(\phi, \alpha_i', \beta_i')$ for all i where $\mathcal{G}(\phi, \alpha_i', \beta_i')$ is a frame.

```julia
function frame_bounds_grid(
    compute_sums::Function,
    dalpha::Real,
    dbeta::Real,
    dw::Real,
    alpha_max::Real,
    beta_max::Real;
    alpha_min::Real = dalpha,
    beta_min::Real = dbeta,
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    optim_iter_s2::Integer = 0,
    print_progress::Integer = 0,
    T1::Type{<:Real} = Float64,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}, Vector{T1}}
    alpha = uniform_spaced_values(alpha_min, alpha_max, dalpha; T=T1)
    beta = uniform_spaced_values(beta_min, beta_max, dbeta; T=T1)

    return frame_bounds(compute_sums, alpha, beta, dw; B_symmetric,
        ensure_integer_w, tol_M, optim_iter_M, optim_iter_s2, print_progress, T2)
end
```

These are all the functions defined in the file `ghosh_selvan.jl`.

The specific methods related to the bounds found in Chapter 5 has been implemented in **modules/ghosh_selvan_guaranteed_bound.jl**. In the beginning of the file are constants denoting which conjectures to assume and whether to calculate d using multiple values of Ω_2.

6.3 Zibulski-Zeevi method in Julia

We will now look at the Zibulksi-Zeevi method defined in the file **modules/zibulski_zeevi.jl**. We start by computing the Zak-transform $Z_\lambda \phi$ of a function $\phi \in L^2(\mathbb{R})$ as defined in Definition 3.3.4 as

$$(Z_\lambda f)(t,\nu) = \lambda^{1/2} \sum_{k=\lfloor t-b/\lambda \rfloor}^{\lceil t-a/\lambda \rceil} f(\lambda(t-k))e^{2\pi i k\nu}, \quad t,\nu \in \mathbb{R}.$$

where $\lambda > 0$ and we assume that $\operatorname{supp}\phi = [a,b]$, since then $a \le \lambda(t-k) \le b$ implies $t - \frac{a}{\lambda} \le k \le t - \frac{b}{\lambda}$.

The function `compute_zak_transform` takes a function $\phi \in L^2(\mathbb{R})$, its compact support $(a,b) \in \mathbb{R}^2$ where $\operatorname{supp}\phi = [a,b]$, $\lambda \in \mathbb{R}$ where $\lambda > 0$, $\boldsymbol{t} = (t_1, \dots, t_m) \in \mathbb{R}^n$, and $\boldsymbol{v} = (v_1, \dots, v_n) \in \mathbb{R}^m$. It returns the complex matrix

$$\boldsymbol{z} = \begin{bmatrix} z_{11} & \cdots & z_{1n} \\ \vdots & \ddots & \vdots \\ z_{m1} & \cdots & z_{mn} \end{bmatrix} \in \mathbb{C}^{m\times n}$$

where $z_{ij} = (Z_\lambda \phi)(t_i, v_j)$ for all i,j.

```julia
function compute_zak_transform(
    phi::Function,
    compact_support::Tuple{Real, Real},
    lamb::Real,
    t::Vector{T},
    v::Vector{T}
)::Matrix{<:Complex} where T<:Real
    a, b = compact_support
    @assert lamb > 0 && a < b

    kmin = floor(Int, minimum(t) - b/lamb)
    kmax = ceil(Int, maximum(t) - a/lamb)

    z = sqrt(lamb) * sum(phi(lamb * (t .- k)) * exp.((2 * pi * im * k) * v)' for
        k in kmin:kmax)

    @assert size(z) == (length(t), length(v))
    return z
end
```

Here we also use the outer product in `phi(lamb * (t .- k)) * exp.((2 * pi * im * k) * v)'`, where `im` is the imaginary unit in Julia, thus giving us an outer product matrix of complex numbers.

Now we can compute the Zibulski-Zeevi matrix defined in Definition 3.3.5:

$$\Phi^{\phi}(t_i, v_j) = p^{-1/2}\left[(Z_{\frac{1}{\beta}}\phi)\left(t_i - \ell\frac{p}{q}, v_j + \frac{k}{p}\right)\right]_{k\in\mathbb{Z}_p;\ \ell\in\mathbb{Z}_q}, \quad \text{for a.e } t_i, v_j \in \mathbb{R}.$$

where $\alpha\beta = \frac{p}{q} \in \mathbb{Q}$.

The function `compute_zibulsk_zeevi_matrix` takes a function ϕ with compact support $(a, b) \in \mathbb{R}^2$ where $\operatorname{supp}\phi = [a, b]$, $\beta \in \mathbb{R}$, $p, q \in \mathbb{Z}_{>0}$ such that $\alpha\beta = \frac{p}{q}$, $\boldsymbol{t} = (t_1, \ldots, t_m)$, and $\boldsymbol{v} = (v_1, \ldots, v_n)$. It returns the complex 4D array

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Phi}_{11} & \ldots & \boldsymbol{\Phi}_{1n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{\Phi}_{m1} & \ldots & \boldsymbol{\Phi}_{mn} \end{bmatrix} \in \mathbb{C}^{m\times n\times p\times q}$$

where $\boldsymbol{\Phi}_{i,j} = \Phi^{\phi}(t_i, v_j) \in \mathbb{C}^{p\times q}$ for all i, j.

```julia
function compute_zibulsk_zeevi_matrix(
    phi::Function,
    compact_support::Tuple{Real, Real},
    beta::Real,
    p::Integer,
    q::Integer,
    t::Vector{T},
    v::Vector{T}
)::Array{<:Complex, 4} where T<:Real
    @assert 0 < p < q && beta > 0

    Phi = stack([compute_zak_transform(phi, compact_support, 1/beta, t .- l *
        p/q, v .+ k/p) for k in 0:p-1, l in 0:q-1] / sqrt(p))

    @assert size(Phi) == (length(t), length(v), p, q)
    return Phi
end
```

The next function is based on Theorem 3.3.6, where we find the frame bounds:

$$A = \operatorname*{ess\,inf}_{t,\nu\in[0,1[} \sigma_r(\Phi^{\phi}(t,\nu))^2 \qquad \text{and} \qquad B = \operatorname*{ess\,sup}_{t,\nu\in[0,1[} \sigma_1(\Phi^{\phi}(t,\nu))^2.$$

Note that $\Phi^{\phi}(t, \nu)$ is 1-periodic on both t and ν, thus we can use unbounded optimization instead of bounded optimization.

The function `compute_frame_bounds` takes:

- A function $\phi \in L^2(\mathbb{R})$,
- Its compact support $(a, b) \in \mathbb{R}^2$ where $\operatorname{supp}\phi = [a, b]$,
- $\alpha, \beta \in \mathbb{R}_{>0}$,
- $p, q \in \mathbb{Z}_{>0}$ such that $\alpha\beta = \frac{p}{q} < 1$ and $\gcd(p, q) = 1$,
- $\boldsymbol{t} = (t_1, \ldots, t_m) \in \mathbb{R}^m$,
- $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$,
- `optim_iter` which is the number of iterations the optimizer runs.

It returns the frame bounds $A \in \mathbb{R}$ and $B \in \mathbb{R}$.

```
function compute_frame_bounds(
    phi::Function,
    compact_support::Tuple{Real, Real},
    alpha::T1,
    beta::T1,
    p::Integer,
    q::Integer,
    t::Vector{T2},
    v::Vector{T2};
    optim_iter::Integer = 0
)::Tuple{T1, T1} where {T1<:Real, T2<:Real}
    @assert abs(alpha*beta - p/q) < 1e-10 && optim_iter >= 0

    Phi = compute_zibulsk_zeevi_matrix(phi, compact_support, beta, p, q, t, v)
    S = map(A -> svdvals(A), Phi[i, j, :, :] for i in axes(Phi, 1), j in
        axes(Phi, 2))
    (A, idx_A) = findmin(map(v -> v[end], S))
    (B, idx_B) = findmax(map(v -> v[1], S))

    if optim_iter > 0
        f(x) = svdvals(compute_zibulsk_zeevi_matrix(phi, compact_support, beta,
            p, q, [x[1]], [x[2]])[1, 1, :, :])

        A = minimize_unbounded(x -> f(x) |> last, [t[idx_A[1]], v[idx_A[2]]], A,
            optim_iter)
        B = maximize_unbounded(x -> f(x) |> first, [t[idx_B[1]], v[idx_B[2]]], B,
            optim_iter)
    end

    @assert size(S) == (length(t), length(v))
    return T1(A^2), T1(B^2)
end
```

The function computes $\mathbf{\Phi} \in \mathbb{C}^{m\times n\times p\times q}$ and computes the SVD-values for each $p \times q$ matrix in the 4D array $\mathbf{\Phi}$. We then find the smallest and largest of the computed SVD-values and if `optim_iter` > 0, we minimize and maximize the function computing the SVD-values with the library `Optim` to get an even smaller and even larger SVD-value.

The function `frame_bounds` takes:

- A function ϕ,
- Its compact support $(a, b) \in \mathbb{R}^2$ where $\operatorname{supp}\phi = [a, b]$,
- A vector of reduced fractions $\boldsymbol{r} = (r_1, \dots, r_\ell) \in \mathbb{Q}^\ell$ such that $r_i < 1$ for all i,
- $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{R}^n$,
- $\boldsymbol{\beta} = (\alpha_1, \dots, \alpha_m) \in \mathbb{R}^m$,
- d_t which specifies $t_{i+1} - t_i \leq d_t$ for all i,
- d_v which specifies $v_{i+1} - v_i \leq d_v$ for all i,
- $\alpha_{\max} \in \mathbb{R} \cup \{\infty\}$,
- $\beta_{\max} \in \mathbb{R} \cup \{\infty\}$,
- $\alpha_{\min} \in \mathbb{R}$,
- $\beta_{\min} \in \mathbb{R}$,
- `optim_iter` which is the number of iterations the optimizer runs for,

- `is_not_frame` is an optionally provided function $\mathbb{R}^2 \times \mathbb{Z}^2 \to \{\text{false}, \text{true}\}$ which takes (α, β, p, q) as input and return true if we know these parameters cannot be a frame and false otherwise. Based on Theorem 3.3.8 we have implemented a function called `is_not_frame_rationally_oversampled_bspline` in `utils.jl`, which can be used for the B-spline specific case,
- `print_progess` which specifies if progress should be printed,
- `T2` which is the type of floating point number used for calculations.

It returns $\boldsymbol{A} = (A_1, \ldots, A_k), \boldsymbol{B} = (B_1, \ldots, B_k)$, $\boldsymbol{\alpha}' = (\alpha'_1, \ldots, \alpha'_k)$, and $\boldsymbol{\beta}' = (\beta'_1, \ldots, \beta'_k)$ where A_i and B_i are the frame bounds for $\mathcal{G}(\phi, \alpha'_i, \beta'_i)$ for all i.

```julia
function frame_bounds(
    phi::Function,
    compact_support::Tuple{Real, Real},
    fractions::Vector{<:Rational},
    alpha::Vector{T1},
    beta::Vector{T1},
    dt::Real,
    dv::Real;
    alpha_max::Real = Inf,
    beta_max::Real = Inf,
    alpha_min::Real = 0,
    beta_min::Real = 0,
    optim_iter::Integer = 0,
    is_not_frame::Union{Function, Nothing} = nothing,
    print_progress::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}, Vector{T1}} where T1<:Real
    @assert print_progress >= 0 && 0 <= alpha_min <= alpha_max && 0 <= beta_min
        <= beta_max && dt > 0 && dv > 0

    A = T1[]
    B = T1[]
    alpha_new = T1[]
    beta_new = T1[]

    t = uniform_spaced_values(0, 1-dt, dt; T=T2)
    v = uniform_spaced_values(0, 1-dv, dv; T=T2)

    K = length(fractions)
    for (k, frac) in enumerate(fractions)
        pk, qk = numerator(frac), denominator(frac)

        if print_progress > 0
            println("Iteration (p, q) = ($pk, $qk) ($k out of $K)")
        end

        alpha_frac = (pk / qk) ./ beta
        beta_frac = (pk / qk) ./ alpha

        alpha_beta = vcat(collect(zip(alpha_frac, beta)), collect(zip(alpha,
            beta_frac)))
        filter!(tuple -> alpha_min < tuple[1] < alpha_max && beta_min < tuple[2]
            < beta_max, alpha_beta)

        J = length(alpha_beta)
        for (j, (alpha_j, beta_j)) in enumerate(alpha_beta)
            if print_progress > 1
                println(" Iteration (alpha, beta) = ($alpha_j, $beta_j) ($j out of
                    $J)")
            end

            if !isnothing(is_not_frame) && is_not_frame(alpha_j, beta_j, pk, qk)
                Aj, Bj = NaN, NaN
            else
                Aj, Bj = compute_frame_bounds(phi, compact_support, alpha_j,
                    beta_j, pk, qk, t, v; optim_iter)
            end

            push!(A, Aj)
            push!(B, Bj)
            push!(alpha_new, alpha_j)
            push!(beta_new, beta_j)
        end
    end

    @assert length(A) == length(B) == length(alpha_new) == length(beta_new)
    return A, B, alpha_new, beta_new
end
```

In the outer for-loop we go through each reduced fraction $r_i = \frac{p_i}{q_i} \in \mathbb{Q}$ for all i. For each reduced fraction $r_i = \frac{p_i}{q_i}$ we go through each α_j and computes its corresponding $\beta_j'' = \frac{p_i}{q_i \alpha_j}$ and likewise for each β_k we compute its corresponding $\alpha_k'' = \frac{p_i}{q_i \beta_k}$. The pairs (α_j, β_j'') and (α_k'', β_k) are filtered such that only pairs (α_j, β_j'') where $\alpha_{\text{min}} < \alpha_j < \alpha_{\text{max}}$ and $\beta_{\text{min}} < \beta_j'' < \beta_{\text{max}}$, and (α_k'', β_k) where $\alpha_{\text{min}} < \alpha_k'' < \alpha_{\text{max}}$ and $\beta_{\text{min}} < \beta_k < \beta_{\text{max}}$ are kept. We then check if each pair is not a frame with the function `is_not_frame` if it is given as a parameter else we compute the frame bounds with `compute_frame_bounds` and add them to the return lists.

The next function just runs `frame_bounds` on a regular grid of α and β values. The function `frame_bounds_grid` takes:

- A function ϕ,
- Its compact support $(a, b) \in \mathbb{R}^2$ where $\operatorname{supp} \phi = [a, b]$,
- A vector of reduced fractions $\boldsymbol{r} = (r_1, \dots, r_\ell) \in \mathbb{Q}^\ell$ such that $r_i < 1$ for all i,
- d_α which specifies, $\alpha_{i+1} - \alpha_i \leq d_\alpha$ for all i,
- d_β which specifies $\beta_{i+1} - \beta_i \leq d_\beta$ for all i,
- d_t which specifies $t_{i+1} - t_i \leq d_t$ for all i,
- d_v which specifies $v_{i+1} - v_i \leq d_v$ for all i,
- $\alpha_{\text{max}} \in \mathbb{R}$,
- $\beta_{\text{max}} \in \mathbb{R}$,
- $\alpha_{\text{min}} \in \mathbb{R}$,
- $\beta_{\text{min}} \in \mathbb{R}$,
- `optim_iter` which is the number of iterations the optimizer runs,
- `is_not_frame` is an optionally provided function $\mathbb{R}^2 \times \mathbb{Q}^2 \to \{\text{false}, \text{true}\}$ which takes (α, β, p, q) as input and return true if we know these parameters cannot be a frame and false otherwise. Based on Theorem 3.3.8 we have implemented a function called `is_not_frame_rationally_oversampled_bspline` in `utils.jl`, which can be used for the B-spline specific case,

- `print_progess` which specifies if progress should be printed,
- `T1` which is the type of floating point number that is returned,
- `T2` which is the type of floating point number used for calculations.

It returns $\boldsymbol{A} = (A_1, \dots, A_k)$, $\boldsymbol{B} = (B_1, \dots, B_k)$, $\boldsymbol{\alpha}' = (\alpha'_1, \dots, \alpha'_k)$, and $\boldsymbol{\beta}' = (\beta'_1, \dots, \beta'_k)$ where A_i and B_i are the frame bounds for $\mathcal{G}(\phi, \alpha'_i, \beta'_i)$ for all i.

```julia
function frame_bounds_grid(
    phi::Function,
    compact_support::Tuple{Real, Real},
    fractions::Vector{<:Rational},
    dalpha::Real,
    dbeta::Real,
    dt::Real,
    dv::Real,
    alpha_max::Real,
    beta_max::Real;
    alpha_min::Real = 0,
    beta_min::Real = 0,
    optim_iter::Integer = 0,
    is_not_frame::Union{Function, Nothing} = nothing,
    print_progress::Integer = 0,
    T1::Type{<:Real} = Float64,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{<:Real}, Vector{<:Real}, Vector{<:Real}, Vector{<:Real}}
    alpha = uniform_spaced_values(alpha_min > 0 ? alpha_min : dalpha, alpha_max,
        dalpha; T=T1)
    beta = uniform_spaced_values(beta_min > 0 ? beta_min : dbeta, beta_max,
        dbeta; T=T1)

    return frame_bounds(phi, compact_support, fractions, alpha, beta, dt, dv;
        alpha_max, beta_max, alpha_min, beta_min, optim_iter, is_not_frame,
        print_progress, T2)
end
```

As mentioned before, the code for the Lyubarskii-Nes method is similar to the code for the Zibulski-Zeevi method, so we will therefore not go through the code for that method for the sake of conciseness. The Lyubarskii-Nes method is based on Corollary 3.3.7 and cannot say anything about the frame bounds, but it can estimate the frame set. In Julia the `rank` function computes the rank of a matrix $\boldsymbol{A}$ by computing the SVD of $\boldsymbol{A}$. It then checks how many singular values are above $c \cdot \sigma_1(\boldsymbol{A})$ for some small constant $c > 0$ where $\sigma_1(\boldsymbol{A})$ is the largest singular value of $\boldsymbol{A}$. To check if $\boldsymbol{A}$ does not have full rank, we therefore check that $\frac{\sigma_p(\boldsymbol{A})}{\sigma_1(\boldsymbol{A})} > c$ where $\sigma_p(\boldsymbol{A})$ is the smallest singular value of $\boldsymbol{A}$. The Lyubarskii-Nes method therefore returns the smallest "SVD-ratio" $\frac{\sigma_p(\boldsymbol{A})}{\sigma_1(\boldsymbol{A})}$ of the matrix

$$\boldsymbol{A} = \left[\sum_{n \in \mathbb{Z}} \phi(t + \alpha\ell - \alpha q n + k/\beta) e^{2\pi i n \alpha q \nu} \right]_{k \in \mathbb{Z}_p;\ \ell \in \mathbb{Z}_q},$$

on $(t, \nu) \in [0, \alpha/p[\times [0, 1/\alpha[$. The Lyubarskii-Nes method computes on the same α, β, p, and q values as in the Zibulski-Zeevi method if the same parameters are given.

Chapter 7

Numerical results

We use discretization to approximate the extrema of continuous functions and floating-point numbers to approximate real numbers, which means that our computed frame bounds only approximate the theoretical frame bounds of the Ghosh-Selvan method and Zibulski-Zeevi method.

If we compare the optimal frame bounds and the frame bounds from the Ghosh-Selvan and Zibulski-Zeevi methods, we get the following inequalities:

$$A^{\mathrm{GS}} \leq A^{\mathrm{OP}} = A^{\mathrm{ZZ}} \leq A^{\mathrm{ZZ}}_{\mathrm{num}} \leq B^{\mathrm{ZZ}}_{\mathrm{num}} \leq B^{\mathrm{OP}} = B^{\mathrm{ZZ}} \leq B^{\mathrm{GS}}$$

and

$$A^{\mathrm{GS}} \leq A^{\mathrm{GS}}_{\mathrm{num}} \leq B^{\mathrm{GS}}_{\mathrm{num}} \leq B^{\mathrm{GS}}$$

where A^{OP} and B^{OP} denotes the optimal frame bounds, A^{ZZ} and B^{ZZ} denotes the frame bounds obtained using the Zibulski-Zeevi method, A^{GS} and B^{GS} denotes the frame bounds obtained using the Ghosh-Selvan method, and the subscript X_{num} denotes the numerical estimation for the respective frame bound method.

If we assume $A^{GS} < A^{OP}$, $B^{OP} < B^{GS}$, and our discretization is sufficiently fine we get the inequalities:

$$A^{\mathrm{GS}} \leq A^{\mathrm{GS}}_{\mathrm{num}} \leq A^{\mathrm{OP}} = A^{\mathrm{ZZ}} \leq A^{\mathrm{ZZ}}_{\mathrm{num}} \leq B^{\mathrm{ZZ}}_{\mathrm{num}} \leq B^{\mathrm{OP}} = B^{\mathrm{ZZ}} \leq B^{\mathrm{GS}}_{\mathrm{num}} \leq B^{\mathrm{GS}}$$

and thus $A^{\mathrm{OP}} \in [A^{\mathrm{GS}}_{\mathrm{num}}, A^{\mathrm{ZZ}}_{\mathrm{num}}]$ and $B^{\mathrm{OP}} \in [B^{\mathrm{ZZ}}_{\mathrm{num}}, B^{\mathrm{GS}}_{\mathrm{num}}]$.

This is also what we observe when we plot the frame bounds along a hyperbola.

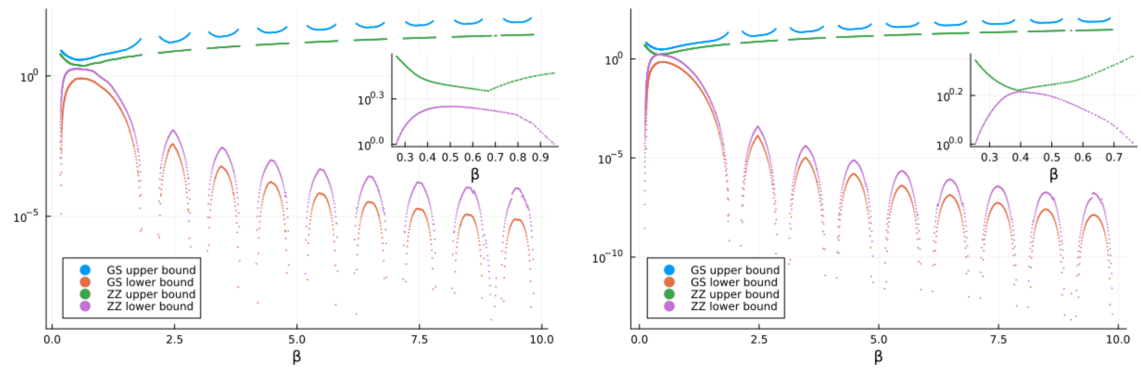

Figure 7.1: The frame bounds for Q_2 and Q_3 along the hyperbola $\alpha\beta = \frac{1}{3}$. Note that the Zibulski-Zeevi frame bounds do not overlap. The plots are generated in **scripts/bounds_hyperbola.jl**.

7.1 B-spline

We have two ways of computing the frame bounds and the corresponding frame set with the Ghosh-Selvan method for the B-spline. One where we use the B-spline specific case for computing $B_{Q_m,1/\beta}$ from Theorem 4.2.4, and one where we use the generic method from Definition 4.1.1 where $\phi = Q_m$. In Julia, this corresponds to using `compute_sums_bspline` and `compute_sums_generic`, respectively.

For the generic method, we define an arbitrarily large compact support for $\widehat{Q}_m$ and assume that values outside the compact support are sufficiently close to zero not to significantly change the frame bounds calculated by the Ghosh-Selvan method.

The B-spline specific case has fewer assumptions and should therefore theoretically lead to better results. Still, we see numerically that $\widehat{Q}_m$ is easier to compute and gives fewer numerical errors compared to computing Q_m, using the formulae in Proposition 2.3.3. The B-spline specific case for computing $B_{Q_m,1/\beta}$ from Theorem 4.2.4 becomes numerically unstable for large β and m. This is because the sums go from 1 to $\lceil m\beta \rceil$ and they sum over a lot of large values, that are numerically imprecise when represented as floating-point numbers.

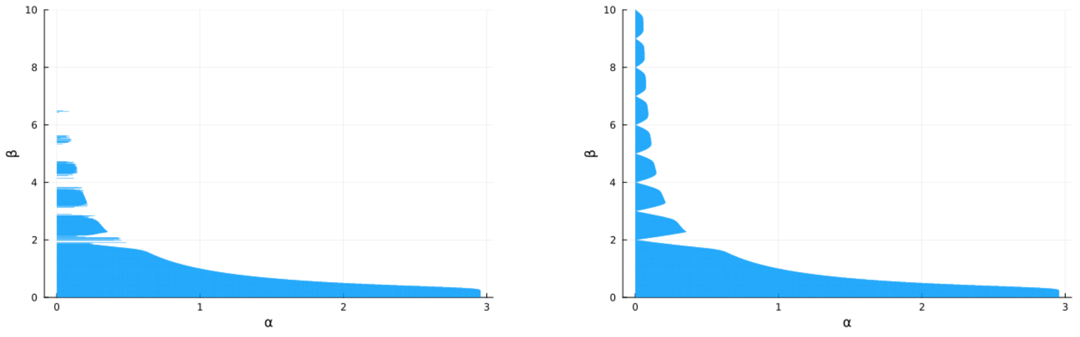

Figure 7.2: The computed frame set of Q_5 where the right side is computed with `compute_sums_bspline` and left side is computed with `compute_sums_generic` with a compact support of $[-200, 200]$. Both are using `T2 = Float64` as the type used for computations. The plots are generated in **scripts/gs_bspline_high_order.jl**.

We see that the `compute_sums_bspline` becomes numerically unstable when m or β increases. The amount that m or β can increase before it becomes unstable depends on the floating-point type. In Fig. 7.2 we see that `T2 = Float64` is too small a floating-point type for even modest m and β values.

We, therefore, use `compute_sums_generic` for the following computations with an arbitrarily chosen compact support of $[-200, 200]$. For $|x| \geq 200$ we only have terms less than $\frac{\sin(\pi x)^2}{x^2} \leq \left(\frac{1}{x\pi}\right)^2 \leq 200^{-2}\pi^{-2} \approx 2.5 \cdot 10^{-6}$ which are ignored for Q_2. Furthermore, we use the datatype `T2 = Double64`.

We have computed the frame set for Q_m where $m \in \{2, 3, 4, 5\}$:

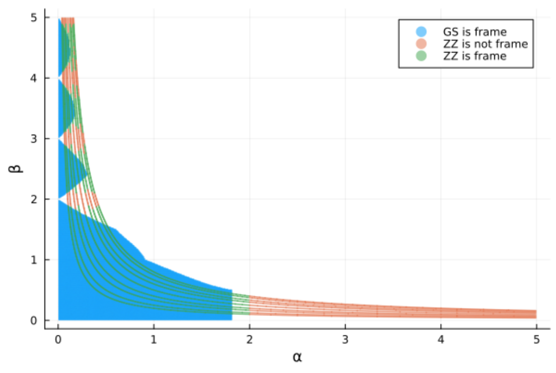

Figure 7.3: The computed frame set for Q_2.

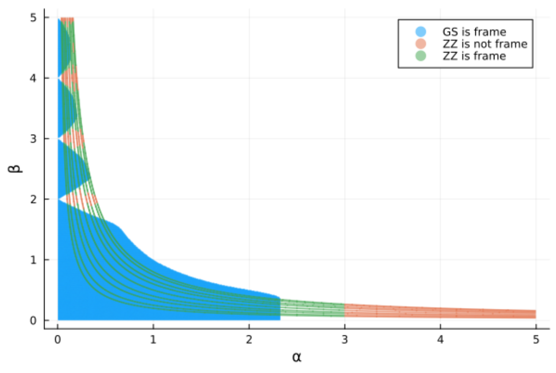

Figure 7.4: The computed frame set for Q_3.

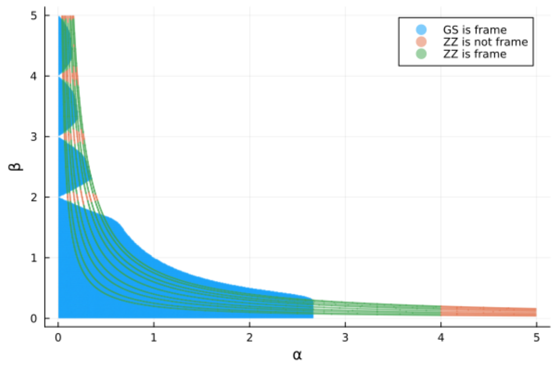

Figure 7.5: The computed frame set for Q_4.

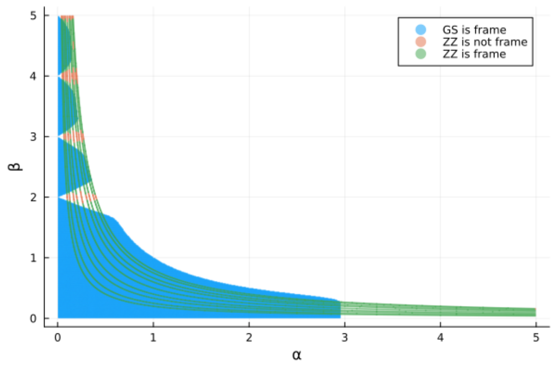

Figure 7.6: The computed frame set for Q_5.

We have also computed the frame bounds. Note that the z-axis is $\log_{10}$ scaled.

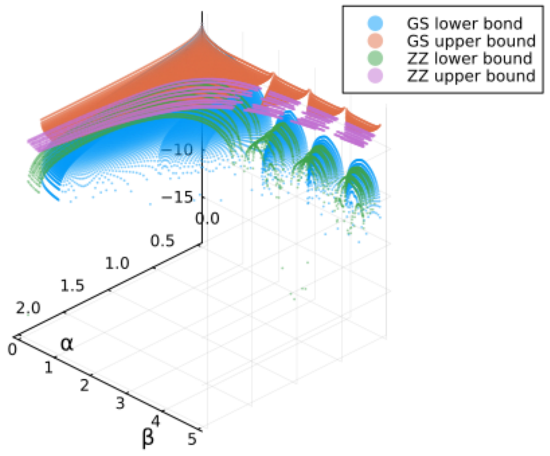

Figure 7.7: The computed frame bounds for Q_2 with $\log_{10}$ scaled z-axis.

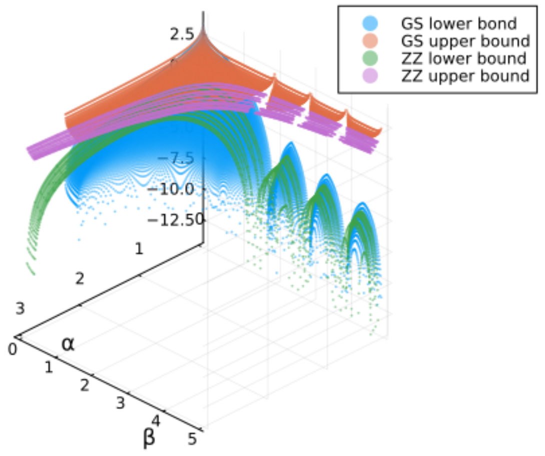

Figure 7.8: The computed frame bounds for Q_3 with $\log_{10}$ scaled z-axis.

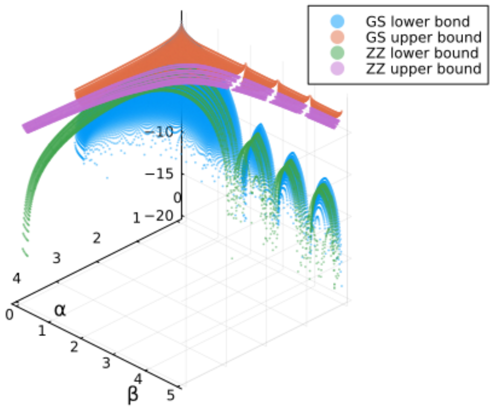

Figure 7.9: The computed frame bounds for Q_4 with $\log_{10}$ scaled z-axis.

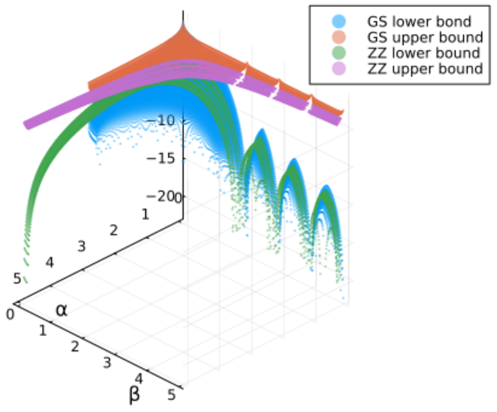

Figure 7.10: The computed frame bounds for Q_5 with $\log_{10}$ scaled z-axis.

The plots are generated in **scripts/gs_zz_bspline.jl**.

7.2 Reparametrization of the frame set

In this section we present two reparametrizations of the frame set which often yield a nicer and more understandable visualization of the frame set. From Theorem 3.2.3 we know that

$$(\alpha,\beta)\in\mathfrak{F}(\phi)\quad \textit{if and only if}\quad (\alpha\delta,\frac{\beta}{\delta})\in\mathfrak{F}(D_\delta\phi) \tag{7.1}$$

where D_δ is the dilation operator $(D_\delta f)(x)=\frac{1}{\sqrt{\delta}}f(\frac{x}{\delta})$. Any two of the three parameters (α,β,δ) can be used to describe the frame set $\mathfrak{F}(D_\delta\phi)$. One choice is to set $\delta=\beta$ and study the frame properties of $\mathcal{G}(D_\beta\phi,\alpha\beta,1)$ since then Eq. (7.1) becomes

$$(\alpha,\beta)\in\mathfrak{F}(\phi)\quad \textit{if and only if}\quad (\alpha\beta,1)\in\mathfrak{F}(D_\beta\phi).$$

This is essentially the choice taken in by Dai and Sun in [8] for the box spline[1].

However, we will follow a related but slightly different reparametrization. Recall from the discussion following Theorem 3.2.2 that $\alpha\beta < 1$ for continuous window functions. Hence, the hyperbolic frame set in the (α, β)-coordinates becomes a box in $(\alpha\beta, \beta)$-coordinates. We know from Theorem 4.1.9 that if $0 < \alpha < \frac{1}{2\sqrt{M_{\phi,1/\beta}}}$ then $\mathcal{G}(\phi, \alpha, \beta)$ forms a frame for $L^2(\mathbb{R})$. Thus, we can simply use the map $\beta \mapsto \frac{\beta}{2\sqrt{M_{\phi,1/\beta}}}$, to plot the outline of the frame set in $(\alpha\beta, \beta)$-coordinates. We will later see that these coordinates for the frame set utilize the $\alpha\beta$-axis, making it possible to visualize the frame set for high values of β, as it would otherwise be too small to see in (α, β)-coordinates.

7.2.1 Lemvig-Nielsen Counterexamples to the B-spline specific case

Lemvig and Nielsen proposed several ways to find values of α and β where $\mathcal{G}(Q_m, \alpha, \beta)$ is not a frame in [17]. We use these counterexamples to get an idea of how close the frame set found using the Ghosh-Selvan method is to these counterexamples. We have used Theorem 3.3.8 and the hyperbolas

$$\alpha\beta = \frac{2k+1}{2(2n+1)}, \quad \text{for } \beta \in \left[\beta_0 - \alpha_0\frac{k-n}{2},\ \beta_0 + \alpha_0\frac{k-n}{2}\right] \tag{7.2}$$

where

$$\alpha_0 = \frac{1}{2n+1}, \quad \beta_0 = \frac{2k+1}{2}, \quad k, n \in \mathbb{N},\ k > n,\ \alpha_0\beta_0 < 1. \tag{7.3}$$

These hyperbolas were conjectured by Lemvig and Nielsen in [17, p. 1450] and proven by Ghosh and Selvan in [12] to be parameters where $\mathcal{G}(Q_2, \alpha, \beta)$ is not a frame.

We have plotted the frame set outline of Q_m along with the counterexamples. Note that we have swapped the axes such that we plot in $(\beta, \alpha\beta)$-coordinates instead of $(\alpha\beta, \beta)$-coordinates.

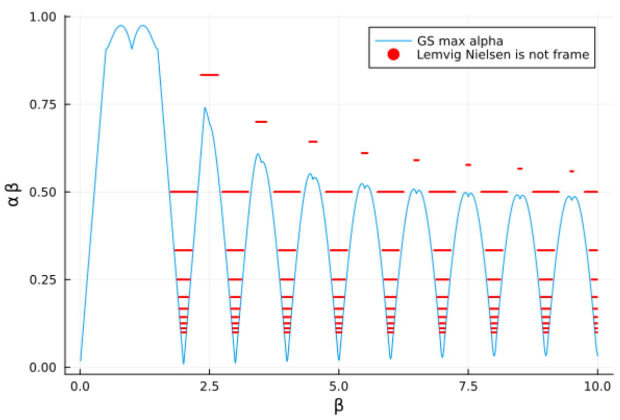

Figure 7.11: The outline of the computed frame set for Q_2 in $(\beta, \alpha\beta)$-coordinates found using the Ghosh-Selvan method. The red lines are some of the non-frame cases from Theorem 3.3.8 and Eq. (7.2).

[1]The B-spline of order $m = 1$.

The plot is generated in **scripts/abc_lemvig_nielsen.jl**.

When $n = k - 1$ in Eq. (7.3) then

$$\alpha_0\beta_0 = \frac{2k+1}{2(2(k-1)+1)} = \frac{2k+1}{4k-2} = \frac{1}{2}\left(\frac{2k}{2k-1} + \frac{1}{2k+1}\right).$$

Thus, we see that

$$\lim_{k\to\infty} \alpha_0\beta_0 = \lim_{k\to\infty} \frac{1}{2}\left(\frac{2k}{2k-1} + \frac{1}{2k+1}\right) = \frac{1}{2}.$$

Although this only holds for Q_2, it gave us the idea to discover whether the tops of the peaks converge toward $\frac{1}{2}$, as this would indicate that the frame set generated by the Ghosh-Selvan method is close to the full frame set.

7.2.2 Peaks

We use the Ghosh-Selvan method to estimate the top of each peak between integers for different B-splines and plot the peaks along the outline of the frame set.

When we do so we get the following plots for Q_m for $m \in \{2, 3, 4, 5\}$:

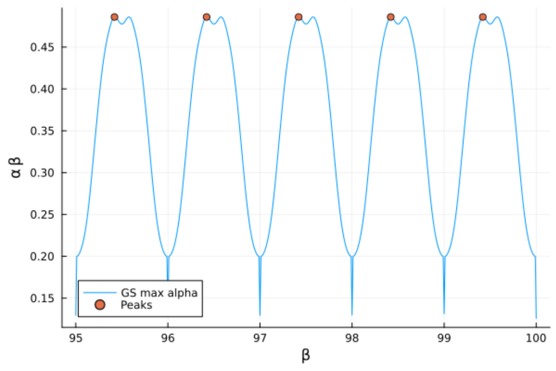

Figure 7.12: The outline of the computed frame set using Ghosh-Selvan for Q_2 in $(\beta, \alpha\beta)$-coordinates.

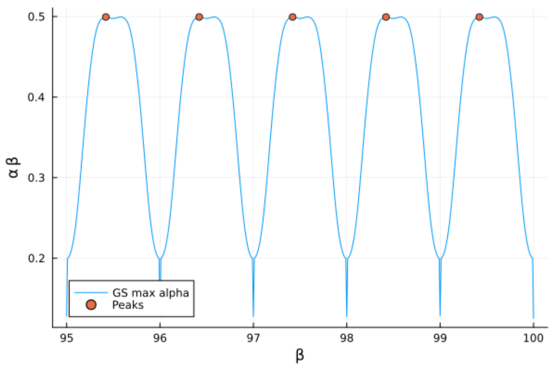

Figure 7.13: The outline of the computed frame set using Ghosh-Selvan for Q_3 in $(\beta, \alpha\beta)$-coordinates.

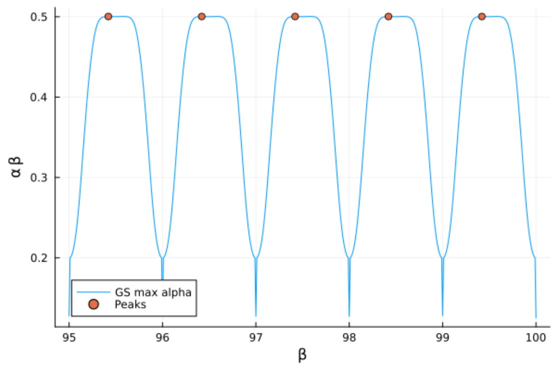

Figure 7.14: The outline of the computed frame set using Ghosh-Selvan for Q_4 in $(\beta, \alpha\beta)$-coordinates.

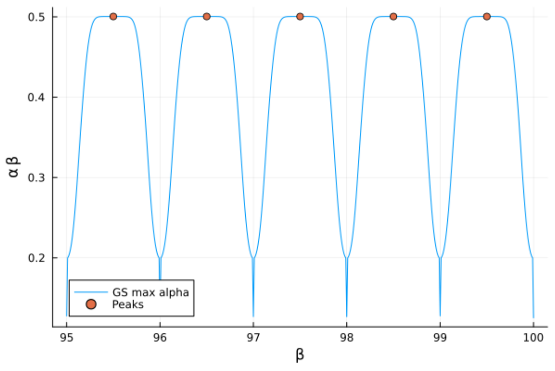

Figure 7.15: The outline of the computed frame set using Ghosh-Selvan for Q_5 in $(\beta, \alpha\beta)$-coordinates.

The above plots are generated in **scripts/abc_gs_bspline_high_beta.jl**.

We have listed the value of each peak in the table below:

$m=2$		$m=3$		$m=4$		$m=5$	
β	$\alpha\beta$	β	$\alpha\beta$	β	$\alpha\beta$	β	$\alpha\beta$
95.42	0.4859169130	95.42	0.4994259453	95.42	0.5003367168	95.50	0.5004851124
96.42	0.4859132015	96.42	0.4994197363	96.42	0.5003294749	96.50	0.5004744457
97.42	0.4859096056	97.42	0.4994137177	97.42	0.5003225541	97.50	0.5004641041
98.42	0.4859061207	98.42	0.4994078816	98.42	0.5003159485	98.50	0.5004540748
99.42	0.4859027422	99.42	0.4994022209	99.42	0.5003096529	99.50	0.5004449995

We see that it looks like the $\alpha\beta$-peaks converge to $\frac{1}{2}$, and they converge slower for increasing m. We therefore plot Q_m for higher $m \in \{10, 20, 30, 40\}$ to see if the peaks still converge to around $\frac{1}{2}$:

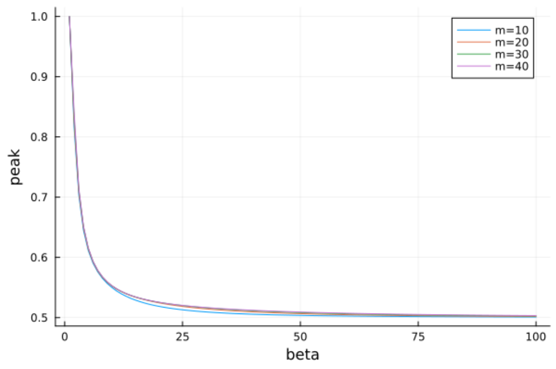

Figure 7.16: The value for each peak between the integers for Q_m where $m \in \{10, 20, 30, 40\}$.

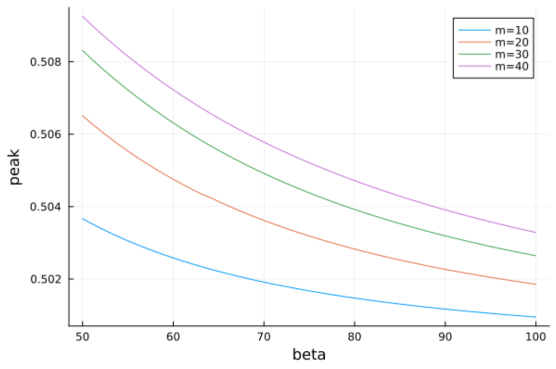

Figure 7.17: Same plot but only showing the last half of the β values.

The peaks are computed in **scripts/peaks.jl** and the plots are generated in **scripts/plot_peak_table**

We see that the $\alpha\beta$-peaks still approach $\frac{1}{2}$, but they converge more gradually for increasing m as expected. From this, we conclude that the Ghosh-Selvan method gives a good approximation of the frame region for Q_m when $\alpha\beta \leq 0.5$.

7.3 Analysis of Hermite functions

In this thesis, we have mainly focused on the B-spline Q_m, but the Ghosh-Selvan and Zibulski-Zeevi methods also work on other types of functions. To give a short example, we have plotted the frame bounds and the corresponding frame set for the Hermite function. We will not give any proofs in this section, but we refer to [13, p. 25], which this section is based upon.

The n'th Hermite function is defined as

$$h_n(x) = (-1)^n (c_n)^{-1/2} e^{\pi x^2} \left(\frac{d^n}{dx^n} e^{-2\pi x^2} \right)$$

where $c_n = (2\pi)^n 2^{n-1/2} n!$ for $n \in \mathbb{Z}_{\geq 0}$. For $n \in \{2,3,4,5\}$, the nth Hermite function is given by:

$$\begin{aligned} h_2(x) &= \frac{2^{\frac{3}{4}}\left(4\pi x^2 - 1\right) e^{-\pi x^2}}{2} \\ h_3(x) &= \frac{\left(4\pi x^3 - 3x\right)\sqrt{3}\, 2^{\frac{3}{4}} \sqrt{\pi}\, e^{-\pi x^2}}{3} \\ h_4(x) &= \frac{2^{\frac{3}{4}}\sqrt{3}\, e^{-\pi x^2}\left(16\pi^2 x^4 - 24\pi x^2 + 3\right)}{12} \\ h_5(x) &= \frac{\left(16\pi^2 x^5 - 40\pi x^3 + 15x\right) 2^{\frac{3}{4}}\sqrt{15}\,\sqrt{\pi}\, e^{-\pi x^2}}{30}. \end{aligned}$$

A plot of the functions can be seen here:

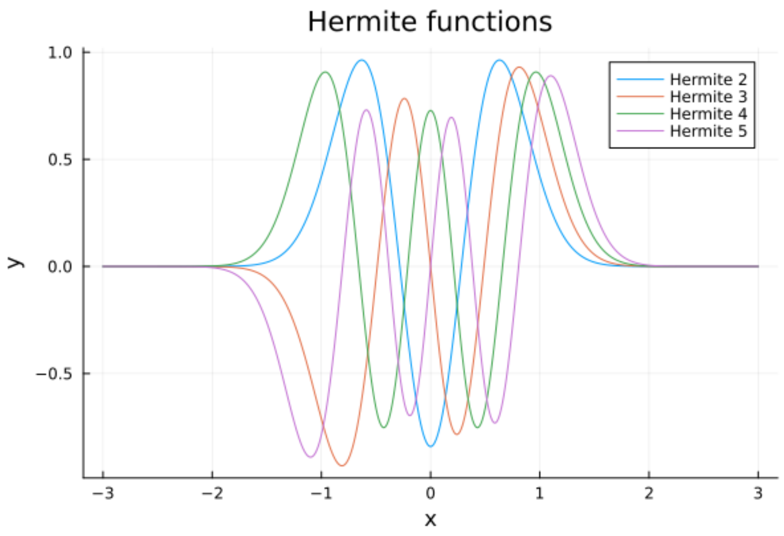

Figure 7.18: The n'th Hermite function h_n for $n \in \{2,3,4,5\}$.

It can be shown that $h_n \in \mathcal{A}$ from Definition 4.1.1 and that

$$\widehat{h_n}(x) = (-1)^n h_n(x), \qquad x \in \mathbb{R}.$$

By Definition 3.1.1 we see that $(\alpha,\beta) \in \mathfrak{F}(h_n)$ if and only if $(\alpha,\beta) \in \mathfrak{F}(\widehat{h_n})$ with the same frame bounds and by Theorem 3.2.4 we know that $(\alpha,\beta) \in \mathfrak{F}(h_n)$ if and only if $(\beta,\alpha) \in \mathfrak{F}(\widehat{h_n})$ with the same frame bounds.

Thus, we can conclude that $(\alpha,\beta) \in \mathfrak{F}(h_n)$ if and only if $(\beta,\alpha) \in \mathfrak{F}(h_n)$ with the same frame bounds.

The Hermite functions does not have compact support, therefore we define an arbitrarily large compact support for the Hermite function and assume that all values outside the compact support are sufficiently close to zero not to significantly change the frame bounds calculated by the Ghosh-Selvan and Zibulski-Zeevi methods. We see that each h_n where $n \in \{2,3,4,5\}$ has the term $e^{-\pi x^2}$ which decays fast, therefore we use the relatively small compact support of $[-10,10]$ in the code.

We have computed an estimate for the frame set of h_n for $n \in \{2,3,4,5\}$:

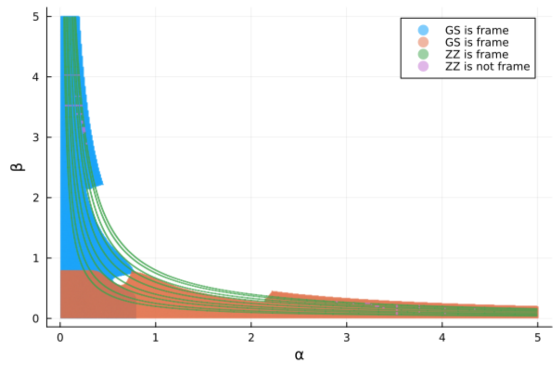

Figure 7.19: The computed frame set for h_2.

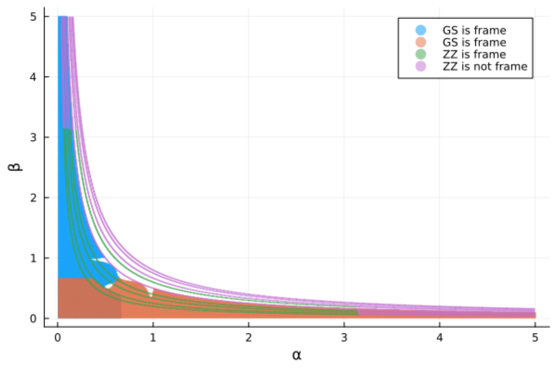

Figure 7.20: The computed frame set for h_3.

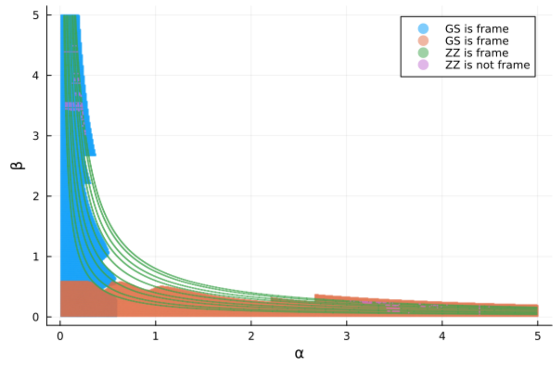

Figure 7.21: The computed frame set for h_4.

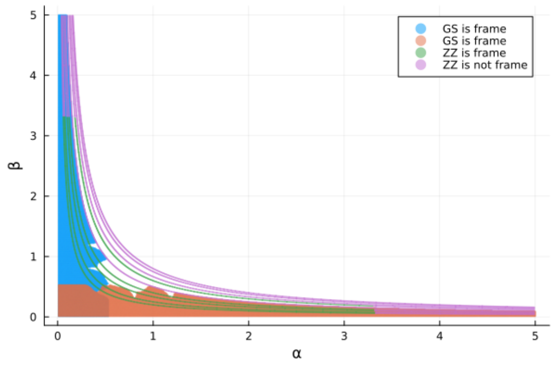

Figure 7.22: The computed frame set for h_5.

We have also computed and estimate for the frame bounds. Note that the z-axis is $\log_{10}$ scaled.

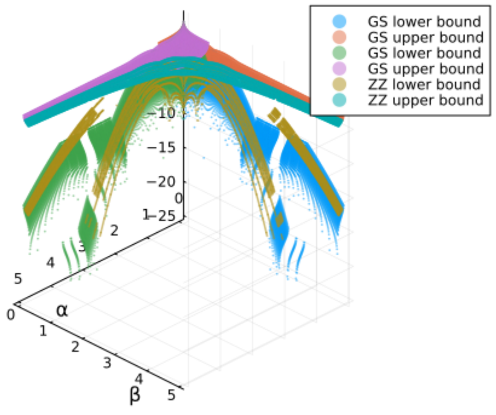

Figure 7.23: The computed frame bounds for h_2 with $\log_{10}$ scaled z-axis.

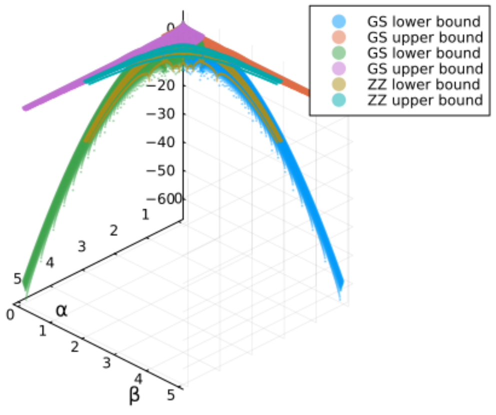

Figure 7.24: The computed frame bounds for h_3 with $\log_{10}$ scaled z-axis.

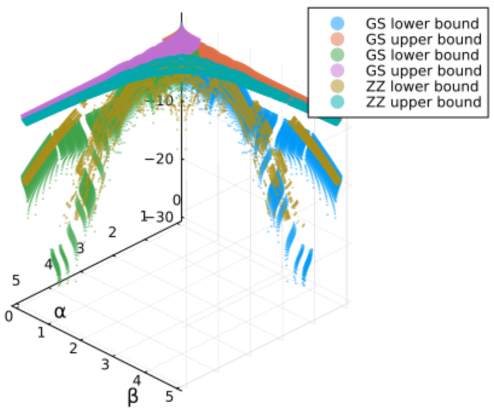

Figure 7.25: The computed frame bounds for h_4 with $\log_{10}$ scaled z-axis.

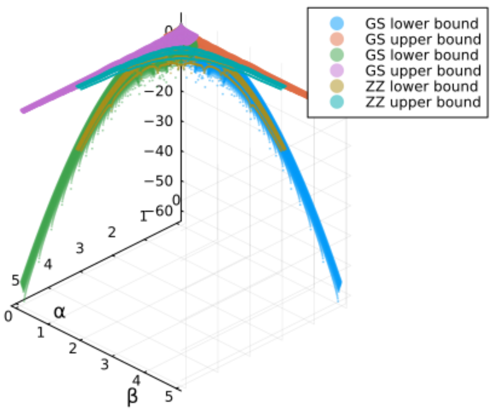

Figure 7.26: The computed frame bounds for h_5 with $\log_{10}$ scaled z-axis.

The above plots are generated in **scripts/gs_zz_hermite.jl**.

The Zibulski-Zeevi method uses the calculated frame bounds to state whether or not $\mathcal{G}(h_n, \alpha, \beta)$ is a frame. This means that due to the numerical errors, we need to set a tolerance for when A is close enough to zero. We set this tolerance as $A > 10^{-20}$, but this value is somewhat arbitrary and depends on the floating-point data type and the computations in the code, therefore it is hard to estimate.

The tolerance of 10^{-20} is set conservatively and could probably be lowered, but this means we can be more sure when the Zibulski-Zeevi method claims a (α, β)-point is a frame compared to when it claims a (α, β)-point is not a frame.

The Ghosh-Selvan method is more numerically robust as the frame set is not computed from the frame bounds, unlike the Zibulski-Zeevi method. This allows for lower values of A, where we can still guarantee that $\mathcal{G}(h_n, \alpha, \beta)$ is a frame.

Chapter 8

Conclusion

We have shown how to correct the Ghosh-Selvan method for computing frame bounds using theory from shift-invariant spaces. Based on the Ghosh-Selvan and Zibulski-Zeevi methods, we were able to develop a toolbox in the programming language Julia, to estimate the frame bounds and the corresponding frame set from these methods.

We have used this toolbox to visualize the computed frame set and computed frame bounds of the B-splines of order $m \geq 2$. In this way we could compare the two methods and found that they did not contradict one another.

One downside of both methods is that they do not have a closed form, which causes discretization and floating-point errors when estimating the frame bounds and frame sets numerically. Thus, we used a simple closed form method for computing obstructions to the frame set for B-splines. From this, we found that the Ghosh-Selvan frame expands almost all the way out to these obstructions.

We also showed that a subset of the frame set found using the Ghosh-Selvan method for B-splines of order $m \geq 2$, can be expressed in a closed-form and in Theorem 5.1.1 we showed the novel result that for any positive non-integer β there exists α such that $\mathcal{G}(Q_m, \alpha, \beta)$ is a frame.

Chapter 9

Appendix

9.1 Code

We have created a GitHub repository for the code. The repository can be found at `https://github.com/TheHarcker/NumericalGaborFrames.jl`.

The following Julia code has a few dependencies. They can be installed by running `julia setup.jl`. The file **utils.jl** is a general utility file that imports the libraries we use and defines some general functions.

Listing 9.1: **setup.jl**

```julia
import Pkg
Pkg.add("Plots")
Pkg.add("LinearAlgebra")
Pkg.add("Plots")
Pkg.add("DoubleFloats")
Pkg.add("Optim")
Pkg.add("NPZ")
Pkg.add("CSV")
```

Listing 9.2: **utils.jl**

```julia
using LinearAlgebra
using Plots
using DoubleFloats
using Optim
using NPZ
import CSV

function uniform_spaced_values(
    x1::Real,
    xn::Real,
    dx::Real;
    T::Type{<:Real} = Float64
)::Vector{T}
    @assert dx > 0 && x1 < xn

    n = ceil(Int, (xn - x1) / dx) + 1
    return collect(T, range(x1, stop=xn, length=n))
end

function bspline(m::Integer, x::T)::T where T<:Real
```

```julia
    @assert m > 0

    y = typeof(x)((-m/2 <= x) & (x <= m/2))
    if m >= 2
        y *= sum((-1)^j * binomial(m, j) * max(x + m/2 - j, 0)^(m - 1) for j in
            0:m) / factorial(m - 1)
    end

    return y
end

function bspline(m::Integer, x::Vector{T})::Vector{T} where T<:Real
    @assert m > 0

    y = typeof(x)((-m/2 .<= x) .& (x .<= m/2))
    if m >= 2
        y .*= sum((-1)^j * binomial(m, j) * max.(x .+ (m/2 - j), 0).^(m - 1) for
            j in 0:m) / factorial(m - 1)
    end

    @assert size(x) == size(y)
    return y
end

function bspline_hat(m::Integer, x::T)::T where T<:Real
    @assert m > 0

    if x == 0
        return 1
    end

    return (sin(pi * x) / (pi * x))^m
end

function bspline_hat(m::Integer, x::Vector{T})::Vector{T} where T<:Real
    return bspline_hat.(m, x)
end

function minimize_unbounded(
    f::Function,
    x0::Vector{T},
    y0::T,
    optim_iter::Integer;
    method::M = LBFGS(),
    print_error::Bool = true,
)::T where {T<:Real, M<:Optim.AbstractOptimizer}
    @assert optim_iter > 0

    try
        y = minimum(optimize(f, x0, method, Optim.Options(iterations =
            optim_iter)))
        return min(y, y0)
    catch e
        if print_error
            println("Optim error: ", e)
        end
        return y0
    end
end

function maximize_unbounded(
    f::Function,
    x0::Vector{T},
    y0::T,
    optim_iter::Integer;
    method::M = LBFGS(),
    print_error::Bool = true,
)::T where {T<:Real, M<:Optim.AbstractOptimizer}
    return -minimize_unbounded(x -> -f(x), x0, -y0, optim_iter; method,
        print_error)
end

function minimize_bounded(
    f::Function,
    a::Vector{T},
    b::Vector{T},
    x0::Vector{T},
    y0::T,
    optim_iter::Integer;
    method::M = LBFGS(),
    print_error::Bool = false,
)::T where {T<:Real, M<:Optim.AbstractOptimizer}
    @assert a < b && optim_iter > 0

    try
        y = minimum(optimize(f, prevfloat.(a), nextfloat.(b), x0, Fminbox(method),
            Optim.Options(iterations = optim_iter, outer_iterations = optim_iter)
        ))
        return min(y, y0)
    catch e
        if print_error
            println("Optim error: ", e)
        end
        return y0
    end
end

function maximize_bounded(
    f::Function,
    a::Vector{T},
    b::Vector{T},
    x0::Vector{T},
    y0::T,
    optim_iter::Integer;
    method::M = LBFGS(),
    print_error::Bool = false,
)::T where {T<:Real, M<:Optim.AbstractOptimizer}
    return -minimize_bounded(x -> -f(x), a, b, x0, -y0, optim_iter; method,
        print_error)
end

function generate_reduced_fractions_below_one(pmax::Integer,
    qmax::Integer)::Vector{Rational}
    @assert pmax > 0 && qmax > 0

    fractions = []
    for p in 1:pmax
        for q in (p+1):qmax
            if gcd(p, q) == 1
                push!(fractions, p // q)
            end
        end
    end

    return fractions
end

function is_gabor_frame(A::Real, B::Real; min_tol::Real = 0, max_tol::Real =
    Inf)::Bool
    return min_tol < A <= B < max_tol
end

function is_gabor_frame(
    A::Vector{<:Real},
    B::Vector{<:Real};
    min_tol::Real = 0,
    max_tol::Real = Inf
)::BitVector
    return is_gabor_frame.(A, B; min_tol, max_tol)
end

function is_not_frame_rationally_oversampled_bspline(
    m::Integer,
)::Function
    @assert m > 0

    return (alpha, beta, p, q) -> beta > 3/2 && abs(beta - round(beta)) * m * q
        <= 1
end

function save_bounds(
    path::AbstractString,
    A::Vector{<:Real},
    B::Vector{<:Real},
    alpha::Vector{<:Real},
    beta::Vector{<:Real}
)
    npzwrite(path, Dict("A" => A, "B" => B, "alpha" => alpha, "beta" => beta))
end

function load_bounds(
    path::AbstractString
)::Tuple{Vector{<:Real}, Vector{<:Real}, Vector{<:Real}, Vector{<:Real}}
    d = npzread(path)
    return d["A"], d["B"], d["alpha"], d["beta"]
end

function save_svd_ratio(
    path::AbstractString,
    svd_ratio::Vector{<:Real},
    alpha::Vector{<:Real},
    beta::Vector{<:Real}
)
    npzwrite(path, Dict("svd_ratio" => svd_ratio, "alpha" => alpha, "beta" =>
        beta))
end

function load_svd_ratios(
    path::AbstractString
)::Tuple{Vector{<:Real}, Vector{<:Real}, Vector{<:Real}}
    d = npzread(path)
    return d["svd_ratio"], d["alpha"], d["beta"]
end

function save_alpha_beta(
    path::AbstractString,
    alpha::Vector{<:Real},
    beta::Vector{<:Real},
)
    npzwrite(path, Dict("alpha" => alpha, "beta" => beta))
end

function load_alpha_beta(
    path::AbstractString
)::Tuple{Vector{<:Real}, Vector{<:Real}}
    d = npzread(path)
    return d["alpha"], d["beta"]
end
```

9.1.1 Modules

This section comprises the module code containing the different methods and is in the `modules/` folder.

Listing 9.3: **modules/ghosh_selvan.jl**

```julia
module GhoshSelvan
export compute_sums_bspline, compute_sums_generic, frame_set_max_alpha,
    frame_bounds_fixed_beta, frame_bounds_grid

include("../utils.jl")

function compute_sums_generic(
    phi_hat::Function,
    compact_support::Tuple{Real, Real},
    beta::Real,
    w::Vector{T}
)::Tuple{Vector{T}, Vector{T}} where T<:Real
    a, b = compact_support
    @assert beta > 0 && a < b

    kmin = floor(Int, (a - maximum(w)) / beta)
    kmax = ceil(Int, (b - minimum(w)) / beta)

    s1 = sum((w .+ k*beta).^2 .* abs2.(phi_hat(w .+ k*beta)) for k in kmin:kmax)
    s2 = sum(abs2.(phi_hat(w .+ k*beta)) for k in kmin:kmax)

    @assert size(w) == size(s1) == size(s2)
    return s1, s2
end

function compute_sums_bspline(
    m::Integer,
    beta::Real,
    w::Vector{T}
)::Tuple{Vector{T}, Vector{T}} where T<:Real
    @assert m > 0 && beta > 0

    n = collect(T, 1:ceil(Int, m * beta))
    x = n ./ beta
    cos_vals = cos.(((2 * pi / beta) .* w) * n’)

    s1 = vec(((bspline(2*m-2, T(0)) - bspline(2*m-2, T(1))) .+ sum(cos_vals .* (2*bspline(2*m-2, x)
        - bspline(2*m-2, x .+ 1) - bspline(2*m-2, x .- 1))’, dims=2)) / (2 * pi^2 * beta))
    s2 = vec((bspline(2*m, T(0)) .+ 2*sum(cos_vals .* bspline(2*m, x)’, dims=2)) / beta)

    @assert size(w) == size(s1) == size(s2)
    return s1, s2
end

function compute_M(
    compute_sums::Function,
    beta::Real,
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{T2, Vector{T2}, Vector{T2}}
    @assert optim_iter_M >= 0

    b = B_symmetric ? beta / 2 : beta
    w = uniform_spaced_values(0, b, dw; T=T2)

    if ensure_integer_w && beta > 1
        w = vcat(w, 1:floor(beta))
    end

    s1, s2 = compute_sums(beta, w)
    (M, idx_M) = findmax(s1 ./ s2)

    if optim_iter_M > 0
        f(w) = compute_sums(beta, w) |> x -> x[1][1] / x[2][1]
        M = maximize_bounded(f, [T2(0)], [T2(b)], [w[idx_M]], M, optim_iter_M)
    end

    if beta^2 >= 4*M + tol_M
        M = NaN
    end

    @assert size(w) == size(s2)
    return M, w, s2
end

function compute_min_max_s2(
    compute_sums::Function,
    beta::Real,
    w::Vector{T2},
    s2::Vector{T2};
    optim_iter_s2::Integer = 0
)::Tuple{T2, T2} where T2<:Real
    @assert optim_iter_s2 >= 0 && size(w) == size(s2)

    (s2_min, idx_s2_min) = findmin(s2)
    (s2_max, idx_s2_max) = findmax(s2)

    if optim_iter_s2 > 0
        f(w) = compute_sums(beta, w)[2][1]
        b = w[end]

        s2_min = minimize_bounded(f, [T2(0)], [T2(b)], [w[idx_s2_min]], s2_min,
            optim_iter_s2)
        s2_max = maximize_bounded(f, [T2(0)], [T2(b)], [w[idx_s2_max]], s2_max,
            optim_iter_s2)
    end

    return s2_min, s2_max
end

function frame_set_max_alpha(
    compute_sums::Function,
    beta::Vector{T1},
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    print_progress::Integer = 0,
    parallelize::Bool = false,
    T2::Type{<:Real} = Float64,
)::Vector{T1} where T1<:Real
    @assert print_progress >= 0

    alpha_max = zeros(length(beta))

    I = length(beta)
    function f(i, beta_i)
        if print_progress > 0
            println("Iteration beta = $(beta_i) ($i out of $I)")
        end

```

```julia
        M, _, _ = compute_M(compute_sums, beta_i, dw; B_symmetric,
            ensure_integer_w, tol_M, optim_iter_M, T2)

        if isfinite(M) && M > 0
            alpha_max[i] = 1/(2*sqrt(M))
        end
    end

    if parallelize
        Threads.@threads for i in axes(beta, 1)
            f(i, beta[i])
        end
    else
        for (i, beta_i) in enumerate(beta)
            f(i, beta_i)
        end
    end

    @assert size(alpha_max) == size(beta)
    return alpha_max
end

function frame_bounds_fixed_beta(
    compute_sums::Function,
    alpha::Vector{T1},
    beta::Real,
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    optim_iter_s2::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}} where T1<:Real
    M, w, s2 = compute_M(compute_sums, beta, dw; B_symmetric, ensure_integer_w,
        tol_M, optim_iter_M, T2)

    if isfinite(M) && M > 0
        alpha = alpha[alpha .< 1/(2*sqrt(M))]
        s2_min, s2_max = compute_min_max_s2(compute_sums, beta, w, s2;
            optim_iter_s2)

        c = 2 * sqrt(M)
        A = Vector{T1}((1 .- c .* alpha).^2 .* s2_min ./ alpha)
        B = Vector{T1}((1 .+ c .* alpha).^2 .* s2_max ./ alpha)
    else
        A, B, alpha = T1[], T1[], T1[]
    end

    @assert size(A) == size(B) == size(alpha)
    return A, B, alpha
end

function frame_bounds(
    compute_sums::Function,
    alpha::Vector{T1},
    beta::Vector{T1},
    dw::Real;
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    optim_iter_s2::Integer = 0,
    print_progress::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}, Vector{T1}} where T1<:Real
    A = T1[]
    B = T1[]
    alpha_new = T1[]
    beta_new = T1[]

    I = length(beta)
    for (i, beta_i) in enumerate(beta)
        if print_progress > 0
            println("Iteration beta = $(beta_i) ($i out of $I)")
        end

        Ai, Bi, alpha_i = frame_bounds_fixed_beta(compute_sums, alpha, beta_i,
            dw; B_symmetric, ensure_integer_w, tol_M, optim_iter_M,
            optim_iter_s2, T2)
        N = length(Ai)

        if N > 0
            append!(A, Ai)
            append!(B, Bi)
            append!(alpha_new, alpha_i)
            append!(beta_new, fill(beta_i, N))
        end
    end

    @assert size(A) == size(B) == size(alpha_new) == size(beta_new)
    return A, B, alpha_new, beta_new
end

function frame_bounds_grid(
    compute_sums::Function,
    dalpha::Real,
    dbeta::Real,
    dw::Real,
    alpha_max::Real,
    beta_max::Real;
    alpha_min::Real = dalpha,
    beta_min::Real = dbeta,
    B_symmetric::Bool = false,
    ensure_integer_w::Bool = true,
    tol_M::Real = 1e-10,
    optim_iter_M::Integer = 0,
    optim_iter_s2::Integer = 0,
    print_progress::Integer = 0,
    T1::Type{<:Real} = Float64,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}, Vector{T1}}
```

```
    alpha = uniform_spaced_values(alpha_min, alpha_max, dalpha; T=T1)
    beta = uniform_spaced_values(beta_min, beta_max, dbeta; T=T1)

    return frame_bounds(compute_sums, alpha, beta, dw; B_symmetric,
        ensure_integer_w, tol_M, optim_iter_M, optim_iter_s2, print_progress, T2)
end

end
```

Listing 9.4: **modules/lyubarskii_nes.jl**

```
module LyubarskiiNes
export frame_set, frame_set_grid

include("../utils.jl")

function compute_transform(
    phi::Function,
    compact_support::Tuple{Real, Real},
    alpha::Real,
    q::Integer,
    t::Vector{T},
    v::Vector{T},
)::Matrix{<:Complex} where T<:Real
    a, b = compact_support
    @assert a < b && alpha > 0 && q > 0

    kmin = floor(Int, minimum((t .- b)/(alpha * q)))
    kmax = ceil(Int, maximum((t .- a)/(alpha * q)))

    z = sum(phi(t .- alpha * q * n) * exp.((2 * pi * im * alpha * q * n) .* v)'
        for n in kmin:kmax)

    @assert (length(t), length(v)) == size(z)
    return z
end

function compute_matrix(
    phi::Function,
    compact_support::Tuple{Real, Real},
    alpha::Real,
    beta::Real,
    p::Integer,
    q::Integer,
    t::Vector{T},
    v::Vector{T},
)::Array{<:Complex, 4} where T<:Real
    @assert 0 < p < q && beta > 0

    Phi = stack([compute_transform(phi, compact_support, alpha, q, t .+ (alpha*l
        + k/beta), v) for k in 0:(p-1), l in 0:(q-1)])

    @assert size(Phi) == (length(t), length(v), p, q)
    return Phi
end

function matrix_svd_ratio(A::Matrix{<:Complex})::Real
```

```
    S = svd(A).S
    return S[end]/S[1]
end

function compute_svd_ratio(
    phi::Function,
    compact_support::Tuple{Real, Real},
    alpha::T1,
    beta::T1,
    p::Integer,
    q::Integer,
    dt::Real,
    dv::Real;
    optim_iter::Integer = 0,
    T2::Type{<:Real} = Float64,
)::T1 where T1<:Real
    @assert abs(alpha*beta - p/q) < 1e-10 && dt > 0 && dv > 0 && optim_iter >= 0

    t = uniform_spaced_values(0, alpha/p, dt; T=T2)
    v = uniform_spaced_values(0, 1/alpha, dv; T=T2)
    Phi = compute_matrix(phi, compact_support, alpha, beta, p, q, t, v)

    R = map(A -> matrix_svd_ratio(A), Phi[i, j, :, :] for i in axes(Phi, 1), j in
        axes(Phi, 2))
    (R_min, idx_R_min) = findmin(R)

    if optim_iter > 0
        f(x) = svdvals(compute_matrix(phi, compact_support, alpha, beta, p, q,
            [x[1]], [x[2]])[1, 1, :, :]) |> S -> S[end]/S[1]
        R_min = minimize_unbounded(f, [t[idx_R_min[1]], v[idx_R_min[2]]], R_min,
            optim_iter)
    end

    @assert size(R) == (length(t), length(v))
    return R_min
end

function frame_set(
    phi::Function,
    compact_support::Tuple{Real, Real},
    fractions::Vector{Rational},
    alpha::Vector{T1},
    beta::Vector{T1},
    dt::Real,
    dv::Real;
    alpha_max::Real = Inf,
    beta_max::Real = Inf,
    alpha_min::Real = 0,
    beta_min::Real = 0,
    optim_iter::Integer = 0,
    is_not_frame::Union{Function, Nothing} = nothing,
    print_progress_layer::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}} where T1<:Real
    @assert 0 <= alpha_min <= alpha_max && 0 <= beta_min < beta_max &&
        print_progress_layer >= 0
```

```julia

    svd_ratio = T1[]
    alpha_new = T1[]
    beta_new = T1[]

    K = length(fractions)
    for (k, frac) in enumerate(fractions)
        pk, qk = numerator(frac), denominator(frac)

        if print_progress_layer > 0
            println("Iteration (p, q) = ($pk, $qk) ($k out of $K)")
        end

        alpha_frac = (pk / qk) ./ beta
        beta_frac = (pk / qk) ./ alpha

        alpha_beta = vcat(collect(zip(alpha_frac, beta)), collect(zip(alpha,
            beta_frac)))
        filter!(tuple -> alpha_min < tuple[1] < alpha_max && beta_min < tuple[2]
            < beta_max, alpha_beta)

        J = length(alpha_beta)
        for (j, (alpha_j, beta_j)) in enumerate(alpha_beta)
            if print_progress_layer > 1
                println(" Iteration (alpha, beta) = ($alpha_j, $beta_j) ($j out of
                    $J)")
            end

            if !isnothing(is_not_frame) && is_not_frame(alpha_j, beta_j, pk, qk)
                svd_ratio_j = NaN
            else
                svd_ratio_j = compute_svd_ratio(phi, compact_support, alpha_j,
                    beta_j, pk, qk, dt, dv; optim_iter, T2)
            end

            push!(svd_ratio, svd_ratio_j)
            push!(alpha_new, alpha_j)
            push!(beta_new, beta_j)
        end
    end

    @assert length(svd_ratio) == length(alpha_new) == length(beta_new)
    return svd_ratio, alpha_new, beta_new
end

function frame_set_grid(
    phi::Function,
    compact_support::Tuple{Real, Real},
    fractions::Vector{Rational},
    dalpha::Real,
    dbeta::Real,
    dt::Real,
    dv::Real,
    alpha_max::Real,
    beta_max::Real;
    alpha_min::Real = 0,
    beta_min::Real = 0,
    optim_iter::Integer = 0,
    is_not_frame::Union{Function, Nothing} = nothing,
    print_progress_layer::Integer = 0,
    T1::Type{<:Real} = Float64,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}}
    alpha = uniform_spaced_values(alpha_min > 0 ? alpha_min : dalpha, alpha_max,
        dalpha; T=T1)
    beta = uniform_spaced_values(beta_min > 0 ? beta_min : dbeta, beta_max,
        dbeta; T=T1)

    return frame_set(phi, compact_support, fractions, alpha, beta, dt, dv;
        alpha_max, beta_max, alpha_min, beta_min, optim_iter, is_not_frame,
        print_progress_layer, T2)
end

end
```

Listing 9.5: **modules/zibulski_zeevi.jl**

```
module ZibulskiZeevi
export frame_bounds, frame_bounds_grid

include("../utils.jl")

function compute_zak_transform(
    phi::Function,
    compact_support::Tuple{Real, Real},
    lamb::Real,
    t::Vector{T},
    v::Vector{T}
)::Matrix{<:Complex} where T<:Real
    a, b = compact_support
    @assert lamb > 0 && a < b

    kmin = floor(Int, minimum(t) - b/lamb)
    kmax = ceil(Int, maximum(t) - a/lamb)

    z = sqrt(lamb) * sum(phi(lamb * (t .- k)) * exp.((2 * pi * im * k) * v)' for
        k in kmin:kmax)

    @assert size(z) == (length(t), length(v))
    return z
end

function compute_zibulsk_zeevi_matrix(
    phi::Function,
    compact_support::Tuple{Real, Real},
    beta::Real,
    p::Integer,
    q::Integer,
    t::Vector{T},
    v::Vector{T}
)::Array{<:Complex, 4} where T<:Real
    @assert 0 < p < q && beta > 0

```

```julia
    Phi = stack([compute_zak_transform(phi, compact_support, 1/beta, t .- l *
        p/q, v .+ k/p) for k in 0:p-1, l in 0:q-1] / sqrt(p))

    @assert size(Phi) == (length(t), length(v), p, q)
    return Phi
end

function compute_frame_bounds(
    phi::Function,
    compact_support::Tuple{Real, Real},
    alpha::T1,
    beta::T1,
    p::Integer,
    q::Integer,
    t::Vector{T2},
    v::Vector{T2};
    optim_iter::Integer = 0
)::Tuple{T1, T1} where {T1<:Real, T2<:Real}
    @assert abs(alpha*beta - p/q) < 1e-10 && optim_iter >= 0

    Phi = compute_zibulsk_zeevi_matrix(phi, compact_support, beta, p, q, t, v)
    S = map(A -> svdvals(A), Phi[i, j, :, :] for i in axes(Phi, 1), j in
        axes(Phi, 2))
    (A, idx_A) = findmin(map(v -> v[end], S))
    (B, idx_B) = findmax(map(v -> v[1], S))

    if optim_iter > 0
        f(x) = svdvals(compute_zibulsk_zeevi_matrix(phi, compact_support, beta,
            p, q, [x[1]], [x[2]])[1, 1, :, :])

        A = minimize_unbounded(x -> f(x) |> last, [t[idx_A[1]], v[idx_A[2]]], A,
            optim_iter)
        B = maximize_unbounded(x -> f(x) |> first, [t[idx_B[1]], v[idx_B[2]]], B,
            optim_iter)
    end

    @assert size(S) == (length(t), length(v))
    return T1(A^2), T1(B^2)
end

function frame_bounds(
    phi::Function,
    compact_support::Tuple{Real, Real},
    fractions::Vector{<:Rational},
    alpha::Vector{T1},
    beta::Vector{T1},
    dt::Real,
    dv::Real;
    alpha_max::Real = Inf,
    beta_max::Real = Inf,
    alpha_min::Real = 0,
    beta_min::Real = 0,
    optim_iter::Integer = 0,
    is_not_frame::Union{Function, Nothing} = nothing,
    print_progress::Integer = 0,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{T1}, Vector{T1}, Vector{T1}, Vector{T1}} where T1<:Real
    @assert print_progress >= 0 && 0 <= alpha_min <= alpha_max && 0 <= beta_min
        <= beta_max && dt > 0 && dv > 0

    A = T1[]
    B = T1[]
    alpha_new = T1[]
    beta_new = T1[]

    t = uniform_spaced_values(0, 1-dt, dt; T=T2)
    v = uniform_spaced_values(0, 1-dv, dv; T=T2)

    K = length(fractions)
    for (k, frac) in enumerate(fractions)
        pk, qk = numerator(frac), denominator(frac)

        if print_progress > 0
            println("Iteration (p, q) = ($pk, $qk) ($k out of $K)")
        end

        alpha_frac = (pk / qk) ./ beta
        beta_frac = (pk / qk) ./ alpha

        alpha_beta = vcat(collect(zip(alpha_frac, beta)), collect(zip(alpha,
            beta_frac)))
        filter!(tuple -> alpha_min < tuple[1] < alpha_max && beta_min < tuple[2]
            < beta_max, alpha_beta)

        J = length(alpha_beta)
        for (j, (alpha_j, beta_j)) in enumerate(alpha_beta)
            if print_progress > 1
                println(" Iteration (alpha, beta) = ($alpha_j, $beta_j) ($j out of
                    $J)")
            end

            if !isnothing(is_not_frame) && is_not_frame(alpha_j, beta_j, pk, qk)
                Aj, Bj = NaN, NaN
            else
                Aj, Bj = compute_frame_bounds(phi, compact_support, alpha_j,
                    beta_j, pk, qk, t, v; optim_iter)
            end

            push!(A, Aj)
            push!(B, Bj)
            push!(alpha_new, alpha_j)
            push!(beta_new, beta_j)
        end
    end

    @assert length(A) == length(B) == length(alpha_new) == length(beta_new)
    return A, B, alpha_new, beta_new
end

function frame_bounds_grid(
    phi::Function,
    compact_support::Tuple{Real, Real},
    fractions::Vector{<:Rational},
    dalpha::Real,
    dbeta::Real,
    dt::Real,
    dv::Real,
    alpha_max::Real,
    beta_max::Real;
    alpha_min::Real = 0,
    beta_min::Real = 0,
    optim_iter::Integer = 0,
    is_not_frame::Union{Function, Nothing} = nothing,
    print_progress::Integer = 0,
    T1::Type{<:Real} = Float64,
    T2::Type{<:Real} = Float64,
)::Tuple{Vector{<:Real}, Vector{<:Real}, Vector{<:Real}, Vector{<:Real}}
    alpha = uniform_spaced_values(alpha_min > 0 ? alpha_min : dalpha, alpha_max,
        dalpha; T=T1)
    beta = uniform_spaced_values(beta_min > 0 ? beta_min : dbeta, beta_max,
        dbeta; T=T1)

    return frame_bounds(phi, compact_support, fractions, alpha, beta, dt, dv;
        alpha_max, beta_max, alpha_min, beta_min, optim_iter, is_not_frame,
        print_progress, T2)
end

end
```

Listing 9.6: **modules/ghosh_selvan_guaranteed_bound.jl**

```julia
module GhoshSelvanGuaranteedBound
    module Constants

    const global use_convex_conjecture = false
    const global use_special_cases_of_d = true
    const global assume_sigma_inf_conjecture = false
    end

    module BasicFunctions
    export nu, sigma, sigma_inf, sigma_sup

    using ..Constants

    function nu(q::T2, m::T2)::T2 where T2<:Integer
        if q == 1 || (!Constants.assume_sigma_inf_conjecture && q <= m / 2)
            return 0
        else
            s = if m >= q
                sum(zeta -> (-1)^zeta * binomial(2 * m, m + q * zeta),
                    1:floor(Int, m / q))
            else
                0
            end
            return binomial(2 * m, m) + 2 * s
        end
    end

    function sigma_inf(q::Integer, m::Integer)::Real
        return nu(q, m) * q / 4^m
    end

    function sigma_sup(q::Integer, m::Integer)::Real
    @assert m >= 2 && q >= 2
    s = if m >= q
        sum(zeta -> binomial(2*m, m+q*zeta), 1:floor(Int, m/q))
        else
            0
        end
    value = q / 4^m * (binomial(2*m, m) + 2*s)
    @assert value >= 1
    return value
    end

    function sigma(q::Integer, m::Integer, w::Real)::Real
        return sum(l -> sin(pi * (w+l/q))^(2*m), 1:q)
    end

    end
    module NonPeriodicBound
    using ..BasicFunctions
    using ..Constants

    function min(a::Vector{<:Real}, b::Real)::Vector{<:Real}
        c = copy(a)
        c[c .> b] .= b
        return c
    end

    function get_p(q::Integer, beta_max::Real)::Vector{<:Integer}
        return filter(p -> gcd(p,q) == 1, q+1:ceil(Int, beta_max * q))
    end

    function get_d_from_bound2(m::Integer, omega2_value::Vector{T2}, d_max::Vector{T2}, beta_values::Vector{T2})::Vector{T2} where T2<:Real
        @assert all(size(beta_values) == size(beta_values))
        omega1_2nd_term = beta_values .^ (2-2*m) * (1 + 1/(2*m-3)) + (beta_values - d_max) .^ (2-2*m) .* (1 .+ (beta_values - d_max)./(beta_values * (2*m-3)))

        a::Vector{<:Real} = (1 .+ (beta_values + d_max)./(beta_values * (2*m - 3)))
        b::Vector{<:Real} = (1 .+ (beta_values - d_max)./(beta_values * (2*m - 3)))
        omega1_2nd_term = (beta_values + d_max) .^ (2-2*m) .* a + (beta_values - d_max) .^ (2 - 2*m) .* b
        difference = omega2_value - omega1_2nd_term
        @assert all(difference .> 0)

        temp_d::Vector{<:Real} = sqrt.(difference) / pi^m
        # For d values above 1/pi we can freely increset them to d_max. If some values are above d_max we need to decrease them down to d_max
        indices = temp_d .>= min(d_max, 1/pi)
        temp_d[indices] = d_max[indices]
        return temp_d
    end

    function get_d_from_bound2(m::Integer, omega2_value::Vector{T2}, d_max::Real,
        beta_values::Vector{T2})::Vector{T2} where T2<:Real
        @assert size(omega2_value) == size(beta_values)
        #omega1_2nd_term = beta_values .^ (2-2*m) * (1 + 1/(2*m-3)) +
            (beta_values - d_max) .^ (2-2*m) .* (1 .+ (beta_values -
            d_max)./(beta_values * (2*m-3)))
        a::Vector{<:Real} = (1 .+ (beta_values .+ d_max) ./ (beta_values * (2*m -
            3)))
        b::Vector{<:Real} = (1 .+ (beta_values .- d_max) ./ (beta_values * (2*m -
            3)))
        omega1_2nd_term = (beta_values .+ d_max) .^ (2-2*m) .* a + (beta_values
            .- d_max) .^ (2 - 2*m) .* b

        difference = omega2_value - omega1_2nd_term
        @assert all(difference .> 0)
        temp_d::Vector{T2} = sqrt.(difference) * (1/pi)^m
        # Increase from 1/pi to d_max
        temp_d[temp_d .> Base.min(T2(d_max), 1/pi)] .= d_max
        return temp_d
    end

    function omega2(p::Vector{<:Integer}, q::Integer, m::Integer,
        w::Vector{<:Real})::Vector{<:Real}
        @assert size(p) == size(w)
        a::Vector{<:Real} = w .^(2-2*m) .* (1 .+ w ./ ((2*m-3)*p))
        b::Vector{<:Real} = (p/q .- w) .^ (2-2*m) .* (1 .+ (p/q .- w) ./
            ((2*m-3)*p))
        return sigma.(q, m, w) .* (a + b)
    end
    function omega2(p::Vector{<:Integer}, q::Integer, m::Integer,
        w::Real)::Vector{<:Real}
        a::Vector{<:Real} = w ^(2-2*m) .* (1 .+ w ./ ((2*m-3)*p))
        b::Vector{<:Real} = (p/q .- w) .^ (2-2*m) .* (1 .+ (p/q .- w) ./
            ((2*m-3)*p))
        return sigma(q, m, w) .* (a + b)
    end

    function find_d_slow_good(
        q::Integer,
        m::Integer,
        p::Vector{<:Integer},
        T2::Type{<:Real} = Float64;
        binary_search_for_d_with_iterations::Integer = 0
        )::Vector{T2}
        @assert all(p .>=2)

        best_d::Vector{<:Real} = zeros(size(p))

        adjustment::Vector{<:Integer} = 1 .- p.*q .% 2
        p1 = p - adjustment
        omega2_value = omega2(p, q, m, p1/(2*q))
        temp_d::Vector{T2} = get_d_from_bound2(m, omega2_value, p1/(2*q), p / q)
        best_d = max.(temp_d, best_d)
        if Constants.use_special_cases_of_d
```

```
            beta_values = p / q

            if q % 2 == 0
                # w = 1
                indices = p .>= sqrt(7)*q
                omega2_value = omega2(p[indices], q, m, 1)
                temp_d = zeros(size(best_d))
                temp_d[indices] = get_d_from_bound2(m, omega2_value, 1,
                    beta_values[indices])
                best_d = max.(best_d, temp_d)
                # w = 1/q
                @assert all(p .>= 2)
                omega2_value = omega2(p, q, m, 1/q)
                temp_d = get_d_from_bound2(m, omega2_value, 1/q, beta_values)
                best_d = max.(best_d, temp_d)
            else
                # w = 1 - 1/(2q)
                indices = p .>= sqrt(7)*q
                omega2_value = omega2(p[indices], q, m, 1 - 1/(2*q))

                temp_d = zeros(size(best_d))
                temp_d[indices] = get_d_from_bound2(m, omega2_value, 1 - 1/(2*q),
                    beta_values[indices])
                best_d = max.(best_d, temp_d)
                # w = 1/(2*q)
                @assert all(p .>= 2)
                omega2_value = omega2(p, q, m, 1/(2*q))
                best_d = max.(best_d, get_d_from_bound2(m, omega2_value, 1/(2*q),
                    beta_values))
            end
        end

        if binary_search_for_d_with_iterations > 0
            d_min = best_d
            d_max = -((q % 2) .- p)/(2*q)

            omega2_value = omega2(p, q, m, p1/(2*q))

            iter = 0
            while iter < binary_search_for_d_with_iterations
                iter += 1
                interval_lengths = d_max - d_min
                points_to_test = d_min + interval_lengths / 2

                W = Base.min.(points_to_test .^ 2 * pi ^ (2*m), (1/pi) ^ (2 - 2*m))
                omega_1_value = W + beta_values .^ (2-2*m) .* (1+1/(2*m-3)) .+
                    (beta_values - points_to_test) .^ (2-2*m) .* (1 .+
                    (beta_values - points_to_test) ./ (beta_values * (2*m-3)))
                valid_d = omega2_value .>= omega_1_value
                invalid_d = omega2_value .< omega_1_value
                d_min[valid_d] = points_to_test[valid_d]
                d_max[invalid_d] = points_to_test[invalid_d]
            end
            best_d = d_min
            end
        return best_d
```

```julia
    end

    function B(m::Integer, w::Vector{<:Real}, p::Vector{<:Integer}, q::Integer;
        use_omega_1_as_numerator::Bool = false)::Vector{<:Real}
        @assert size(w) == size(p)
        beta_values = p./q
        omega_2_value = w .^ (2-2*m) + (beta_values - w) .^ (2-2*m) + (w .^
            (3-2*m) + (beta_values - w) .^ (3-2*m))./((2*m-3)*p)
        omega_1_with_max_d = (pi^(2*m-2) .+ beta_values .^ (2-2*m)) * 2^(2*m-1) /
            sigma_inf(q, m)

        numerator::Vector{<:Real} = if use_omega_1_as_numerator
            omega_1_with_max_d else omega_2_value end

        denominator = if Constants.use_convex_conjecture &&
            use_omega_1_as_numerator
           ((w+p-beta_values) .^ (1-2*m) + (p-w) .^ (1-2*m)) ./ ((2*m-1)*p)
        else
           2*(p-beta_values/2).^(1-2*m) ./ ((2*m-1)*p)
        end

        return numerator ./ denominator
    end

    function get_points_non_periodic_bound(
        q::Integer,
        m::Integer,
        beta_max::Integer,
        T2::Type{<:Real} = Float64;
        binary_search_for_d_with_iterations::Integer = 0,
        use_omega_1_as_numerator::Bool = false
        )::Tuple{Vector{T2}, Vector{T2}}

        p = get_p(q, beta_max)
        beta_values = p/q
        if q == 1
           return (zeros(size(beta_values)), beta_values)
        end
        d::Vector{<:Real} = find_d_slow_good(q, m, p;
            binary_search_for_d_with_iterations)
        at_first_point::Vector{<:Real} = B(m, d, p, q; use_omega_1_as_numerator)
        at_second_point::Vector{<:Real} = B(m, beta_values / 2, p, q;
            use_omega_1_as_numerator)
        M::Vector{<:Real} = max.(at_first_point, at_second_point)
        max_alpha_values::Vector{<:Real} = 1 ./ (2*sqrt.(M))
        return (max_alpha_values, beta_values)
    end
    end



    module MarziehBound
        function sinc(w::Vector{T2}, m::Integer)::Vector{T2} where T2<:Real
           return (sin.(pi*w)./(pi*w)) .^ (2*m)
        end

        function get_point_for_beta(beta::T2, m::Integer)::T2 where T2<:Real
            w = collect(LinRange(0, beta / 2, 10000))
            sinc_values = sinc(w, m) + sinc(w .- beta, m)
            infimum::Real = minimum(x->isnan(x) ? Inf : x, sinc_values)
            return pi/2 * sqrt(2*beta / (1+2*m*beta) * infimum)
        end
        function get_points(beta_values::Vector{T2}, m::Integer)::Vector{T2}
             where T2<:Real
            return get_point_for_beta.(beta_values, m)
        end
    end
end
```

9.1.2 Scripts

The following scripts contains the code we used to make the plots throughout the thesis and are in the `scripts/` folder.

Listing 9.7: **scripts/abc_gs_bspline_high_beta.jl**

```
include("../utils.jl")
include("../modules/ghosh_selvan.jl")

import .GhoshSelvan as gs

compute = true
T1 = Float64
T2 = Double64
optim_iter_M = 20
print_progress = 1

dbeta = 0.01
dw = 0.001
compact_support_hat = (-200, 200)

beta_min = 95
beta_max = 100

function find_alpha_peaks_non_integer(
    alpha::Vector{<:Real},
    beta::Vector{<:Real};
    tol::Real = 1e-3
)::Vector{Int}
    peaks_idx = Vector{Int}()

    beta_min_int = floor(Int, minimum(beta))
    beta_max_int = ceil(Int, maximum(beta))

    for n in beta_min_int:(beta_max_int - 1)
        bools = n + tol .< beta .< n + 1 - tol
        push!(peaks_idx, argmax(alpha .* bools))
    end

    return peaks_idx
end

```

```julia
for m in 2:5
    phi_hat = x -> bspline_hat(m, x)
    compute_sums(beta, w) = gs.compute_sums_generic(phi_hat, compact_support_hat,
        beta, w)

    file_path = "../Data/frame_sets/abc_gs_high_beta_bspline_$m.npz"
    if compute
        beta = uniform_spaced_values(beta_min, beta_max, dbeta; T=T1)
        alpha = gs.frame_set_max_alpha(compute_sums, beta, dw; optim_iter_M,
            print_progress, T2)
        alpha .*= beta

        save_alpha_beta(file_path, alpha, beta)
    else
        alpha, beta = load_alpha_beta(file_path)
    end

    peaks_idx = find_alpha_peaks_non_integer(alpha, beta)
    peaks_alpha = alpha[peaks_idx]
    peaks_beta = beta[peaks_idx]

    println(collect(zip(peaks_beta, peaks_alpha)))

    plt = plot(beta, alpha, label = "GS max alpha")
    scatter!(peaks_beta, peaks_alpha, label = "Peaks")
    xlabel!("\\beta")
    ylabel!("\\alpha \\beta")
    display(plt)
    savefig(plt, "../Figures/julia/abc_gs_high_beta_bspline_$m.png")
end
```

Listing 9.8: **scripts/abc_lemvig_nielsen.jl**

```julia
include("../utils.jl")
include("../modules/ghosh_selvan.jl")

import .GhoshSelvan as gs

compute = true
T1 = Float64
T2 = Double64
optim_iter_M = 20
print_progress = 1

m = 2
phi_hat = x -> bspline_hat(m, x)
compact_support = (-200, 200)
#compute_sums(beta, w) = gs.compute_sums_bspline(m, beta, w)
compute_sums(beta, w) = gs.compute_sums_generic(phi_hat, compact_support, beta,
    w)

fractions = generate_reduced_fractions_below_one(1, 10)
beta_min = 0
beta_max = 10
dbeta = 0.01
dw = 0.001

```

```julia
function lemvig_nielsen(
    m::Integer,
    fractions::Vector{<:Rational},
    alpha::Vector{T1},
    beta::Vector{T1};
    alpha_max::Real = Inf,
    beta_max::Real = Inf,
    alpha_min::Real = 0,
    beta_min::Real = 0,
)::Tuple{BitVector, Vector{T1}, Vector{T1}} where T1<:Real
    f = is_not_frame_rationally_oversampled_bspline(m)
    is_not_frame = []
    alpha_new = T1[]
    beta_new = T1[]

    for frac in fractions
        pk, qk = numerator(frac), denominator(frac)

        alpha_frac = (pk / qk) ./ beta
        beta_frac = (pk / qk) ./ alpha

        alpha_beta = vcat(collect(zip(alpha_frac, beta)), collect(zip(alpha,
            beta_frac)))
        filter!(tuple -> alpha_min < tuple[1] < alpha_max && beta_min < tuple[2]
            < beta_max, alpha_beta)

        for (alpha_j, beta_j) in alpha_beta
            push!(is_not_frame, f(alpha_j, beta_j, pk, qk))
            push!(alpha_new, alpha_j)
            push!(beta_new, beta_j)
        end
    end

    return is_not_frame, alpha_new, beta_new
end

file_path_gs = "../Data/frame_sets/abc_lemvig_nielsen_bspline_$m.npz"
if compute
    beta_gs = uniform_spaced_values(dbeta, beta_max, dbeta; T=T1)
    alpha_gs = gs.frame_set_max_alpha(compute_sums, beta_gs, dw; optim_iter_M,
        print_progress, T2)
    alpha_gs .*= beta_gs

    save_alpha_beta(file_path_gs, alpha_gs, beta_gs)
else
    alpha_gs, beta_gs = load_alpha_beta(file_path_gs)
end

beta_ce = []
alpha_ce = []
if m == 2
    counter_examples = [(2,1), (3,2), (4,3), (5,4), (6,5), (7,6), (8,7), (9,8)]
    for (i, (k, m)) in enumerate(counter_examples)
        a0 = 1/(2*m + 1)
        b0 = (2*k + 1)/2
        append!(beta_ce, uniform_spaced_values(b0- a0*(k-m)/2, b0 + a0*(k-m)/2,
            0.01))
        append!(alpha_ce, (2*k+1)./(2*(2*m+1).*uniform_spaced_values(b0-
            a0*(k-m)/2, b0 + a0*(k-m)/2, 0.01)))
    end
end
alpha_ce .*= beta_ce

is_not_frame, alpha, beta = lemvig_nielsen(m, fractions, T1[], beta_gs; beta_max)
alpha .*= beta

plt = plot(beta_gs, alpha_gs, label = "GS max alpha")
scatter!(beta[is_not_frame], alpha[is_not_frame], markersize=1,
    markerstrokewidth=0, label = "Lemvig Nielsen is not frame", c=:red)
scatter!(beta_ce, alpha_ce, markersize = 1, markerstrokewidth = 0, c=:red, label
    = false)
xlabel!("\\beta")
ylabel!("\\alpha \\beta")
display(plt)
savefig(plt, "../Figures/julia/abc_lemvig_nielsen_bspline_$m.png")
```

Listing 9.9: **scripts/bounds_hyperbola.jl**

```
include("../utils.jl")
include("../modules/ghosh_selvan.jl")
include("../modules/zibulski_zeevi.jl")

import .ZibulskiZeevi as zz
import .GhoshSelvan as gs
using Plots

m = 3

gs_path = "../Data/bspline_order_$(m)_hyperbola_ghosh_selvan_bounds.npz"
zz_path = "../Data/bspline_order_$(m)_hyperbola_zibulski_zeevi_bounds.npz"

T1 = Float64
T2 = Double64
optim_iter_M = 20
optim_iter_s2 = 20
optim_iter = 20
print_progress = 2

B_symmetric = true

dalpha = 0.01
dbeta = 0.01
dw = 0.001

dt = 0.01
dv = 0.01

alpha_max = 5
beta_min = 0
beta_max = 10
fractions = [1//3]

compute = true
```

```julia
if compute
    phi = x -> bspline(m, x)
    compact_support = (-m/2, m/2)
    is_not_frame = nothing
    A_zz, B_zz, alpha_zz, beta_zz = zz.frame_bounds_grid(phi, compact_support,
        fractions, dalpha, dbeta, dt, dv, alpha_max, beta_max; optim_iter,
        is_not_frame, print_progress = print_progress, T1, T2)
    save_bounds(zz_path, A_zz, B_zz, alpha_zz, beta_zz)

    phi_hat(x) = bspline_hat(m, x)
    compact_support = (-200, 200)
    compute_sums(beta, w) = gs.compute_sums_generic(phi_hat, compact_support,
        beta, w)

    I = length(alpha_zz)

    A_gs = T1[]
    B_gs = T1[]
    alpha_gs = T1[]
    beta_gs = T1[]

    for (i, (alpha_i, beta_i)) in enumerate(zip(alpha_zz, beta_zz))
        if print_progress > 0
            println("Iteration (alpha, beta) = ($alpha_i, $beta_i) ($i out of
                $I)")
        end
        Ai, Bi, _= gs.frame_bounds_fixed_beta(compute_sums, [alpha_i], beta_i,
            dw; B_symmetric, optim_iter_M, optim_iter_s2, T2)

        if length(Ai) > 0
            append!(A_gs, Ai)
            append!(B_gs, Bi)
            append!(alpha_gs, alpha_i)
            append!(beta_gs, beta_i)
        end
    end
    save_bounds(gs_path, A_gs, B_gs, alpha_gs, beta_gs)

else
    A_zz, B_zz, alpha_zz, beta_zz = load_bounds(zz_path)
    A_gs, B_gs, alpha_gs, beta_gs = load_bounds(gs_path)
end

is_frame_gs = is_gabor_frame(A_gs, B_gs)
is_frame_zz = is_gabor_frame(A_zz, B_zz, min_tol = 1e-20)

is_frame_subplot_gs = is_gabor_frame(A_gs, B_gs, min_tol = 1e-1)
is_frame_subplot_zz = is_gabor_frame(A_zz, B_zz, min_tol = 1e-1)


### Plot bounds along hyperbola
plt = scatter(beta_gs[is_frame_gs], B_gs[is_frame_gs], markersize=1,
    markerstrokewidth=0, label="GS upper bound", yscale =:log10,
    legend=:bottomleft)
scatter!(beta_gs[is_frame_gs], A_gs[is_frame_gs], markersize=1,
    markerstrokewidth=0, label="GS lower bound")
```

```julia
scatter!(beta_zz[is_frame_zz], B_zz[is_frame_zz], markersize=1,
    markerstrokewidth=0, label="ZZ upper bound")
scatter!(beta_zz[is_frame_zz], A_zz[is_frame_zz], markersize=1,
    markerstrokewidth=0, label="ZZ lower bound", )
scatter!(beta_zz[is_frame_subplot_zz], B_zz[is_frame_subplot_zz], markersize=1,
    markerstrokewidth=0, inset=[(1, bbox(0.5, 0.25, 0.25, 0.25, :bottom,
    :right))], subplot = 1)
scatter!(beta_zz[is_frame_subplot_zz], A_zz[is_frame_subplot_zz], markersize=1,
    markerstrokewidth=0, inset=[(1, bbox(0.5, 0.25, 0.25, 0.25, :bottom,
    :right))], subplot = 1)

xlabel!("\\beta")
ylabel!("Bounds")
display(plt)
savefig(plt, "../Figures/julia/bspline_order_$(m)_hyperbola_bounds.png")
```

Listing 9.10: **scripts/bspline.jl**

```julia
include("../utils.jl")

m = 10
dx = 0.01

plt = plot()
x = uniform_spaced_values(-5, 5, dx)
for m in 1:10
    plot!(x, bspline(m, x), label="B-spline $m", xlabel="x", ylabel="y")
end
display(plt)
savefig(plt, "../Figures/julia/Bsplines.png")

plt = plot()
x = uniform_spaced_values(-10, 10, dx)
for m in 1:10
    plot!(x, bspline_hat(m, x), label="Fourier transform of B-spline $m",
        xlabel="x", ylabel="y")
end
display(plt)
savefig(plt, "../Figures/julia/Bsplines_hat.png")
```

Listing 9.11: **scripts/gs_bspline_case.jl**

```julia
include("../utils.jl")
include("../modules/ghosh_selvan.jl")

using .GhoshSelvan

T1 = Float64
T2 = Double64
print_progress = 1
B_symmetric = true
optim_iter_M = 100
optim_iter_s2 = 100

m = 2
compute_sums = (beta, w) -> compute_sums_bspline(m, beta, w)

alpha_max = 5
```

```julia
beta_max = 10
dalpha = 0.005
dbeta = 0.01
dw = 0.001

compute = true
if compute

    A, B, alpha, beta = frame_bounds_grid(compute_sums, dalpha, dbeta, dw,
        alpha_max, beta_max; optim_iter_M, optim_iter_s2, B_symmetric,
        print_progress, T1, T2)
    save_bounds("../Data/frame_bounds/gs_bspline_case_$m.npz", A, B, alpha, beta)
else
    A, B, alpha, beta = load_bounds("../Data/frame_bounds/gs_bspline_case_$m.npz")
end

is_frame = is_gabor_frame(A, B)
plt = scatter(alpha[is_frame], beta[is_frame], markersize=1,
    markerstrokewidth=0, xlabel="\\alpha", ylabel="\\beta", label="GS frame
    region")
display(plt)
savefig(plt, "../Figures/julia/frame_set_gs_bpsline_case_$m.png")
```

Listing 9.12: **scripts/gs_bspline_high_order.jl**

```julia
include("../utils.jl")
include("../modules/ghosh_selvan.jl")

import .GhoshSelvan as gs

compute = true
T1 = Float64
T2 = Float64
optim_iter_M = 20
optim_iter_s2 = 20
print_progress = 2

m = 5
B_symmetric = true

dalpha = 0.002
dbeta = 0.01
dw = 0.001

alpha_max = 5
beta_min = dbeta
beta_max = 10

# Compute and save or load old
file_path1 = "../Data/frame_sets/frame_set_gs_special_bspline_$m.npz"
file_path2 = "../Data/frame_sets/frame_set_gs_generic_bspline_$m.npz"
if compute
    compute_sums1(beta, w) = gs.compute_sums_bspline(m, beta, w)
    A1, B1, alpha1, beta1 = gs.frame_bounds_grid(compute_sums1, dalpha, dbeta,
        dw, alpha_max, beta_max; beta_min, optim_iter_M, optim_iter_s2,
        B_symmetric, print_progress, T1, T2)

    phi_hat(x) = bspline_hat(m, x)
    compact_support = (-200, 200)
    compute_sums2(beta, w) = gs.compute_sums_generic(phi_hat, compact_support,
        beta, w)
    A2, B2, alpha2, beta2 = gs.frame_bounds_grid(compute_sums2, dalpha, dbeta,
        dw, alpha_max, beta_max; beta_min, optim_iter_M, optim_iter_s2,
        B_symmetric, print_progress, T1, T2)

    save_bounds(file_path1, A1, B1, alpha1, beta1)
    save_bounds(file_path2, A2, B2, alpha2, beta2)
else
    A1, B1, alpha1, beta1 = load_bounds(file_path1)
    A2, B2, alpha2, beta2 = load_bounds(file_path2)
end

is_frame1 = is_gabor_frame(A1, B1)
is_frame2 = is_gabor_frame(A2, B2)

plt = scatter(alpha1[is_frame1], beta1[is_frame1], markersize=0.3,
    markerstrokewidth=0, legend=false)
xlabel!("\\alpha")
ylabel!("\\beta")
ylims!(0, beta_max)
display(plt)
savefig(plt, "../Figures/julia/frame_set_gs_special_bspline_$m.png")

plt = scatter(alpha2[is_frame2], beta2[is_frame2], markersize=0.3,
    markerstrokewidth=0, legend=false)
xlabel!("\\alpha")
ylabel!("\\beta")
ylims!(0, beta_max)
display(plt)
savefig(plt, "../Figures/julia/frame_set_gs_generic_bspline_$m.png")
```

Listing 9.13: **scripts/gs_guaranteed_bound.jl**

```julia

include("../utils.jl")
include("../modules/ghosh_selvan.jl")
include("../modules/ghosh_selvan_guaranteed_bound.jl")
import .GhoshSelvan as gs

function plot_guaranteed_bound(m::Integer, num_q::Integer, alpha_max::Real,
    beta_min::Integer, beta_max::Integer; show_as_ab_b_plot::Bool=false,
    make_plot::Bool = true, binary_search_for_d_with_iterations::Integer = 10)
    @assert num_q > 0 && m > 0 && alpha_max > 0 && beta_max >= beta_min> 0
    T1 = Float64
    alphas_omega2::Vector{T1} = []
    alphas_omega1::Vector{T1} = []
    betas::Vector{T1} = []

    for q in 1:num_q
        (new_non_periodic_alphas_omega2, new_betas) =
            GhoshSelvanGuaranteedBound.NonPeriodicBound.get_points_non_periodic_bound(q,
            m, beta_max; binary_search_for_d_with_iterations)
        alphas_omega2 = vcat(alphas_omega2, new_non_periodic_alphas_omega2)
        (new_non_periodic_alphas_omega1, _) =
            GhoshSelvanGuaranteedBound.NonPeriodicBound.get_points_non_periodic_bound(q,
            m, beta_max; binary_search_for_d_with_iterations,
            use_omega_1_as_numerator=true)
        alphas_omega1 = vcat(alphas_omega1, new_non_periodic_alphas_omega1)

        betas = vcat(betas, new_betas)
        println("Iteration q=", q, " ", q/num_q * 100, "%")
    end

    marzieh_betas = collect(LinRange(beta_min, beta_max, (beta_max - beta_min) *
        1000))
    marzieh_alphas =
        GhoshSelvanGuaranteedBound.MarziehBound.get_points(marzieh_betas, m)

    compute_sums(beta, w) = gs.compute_sums_bspline(m, beta, w)

    dw = 0.001
    gs_betas = collect(LinRange(beta_min, beta_max, (beta_max - beta_min) * 100))
    gs_alphas = gs.frame_set_max_alpha(compute_sums, gs_betas, dw; B_symmetric=
        true, parallelize=true)

    indices = sortperm(gs_betas)
    @assert length(gs_alphas) == length(gs_betas) == length(indices)
    gs_alphas = gs_alphas[indices]
    gs_betas = gs_betas[indices]

    # Show as (ab, b) plot
    if show_as_ab_b_plot
        gs_alphas .*= gs_betas
        alphas_omega1 .*= betas
        alphas_omega2 .*= betas
        marzieh_alphas .*= marzieh_betas
    end

    # Plot bounds for Q_m(x)
    if make_plot
        if show_as_ab_b_plot
            combined_plot = plot(gs_alphas, gs_betas, lw=1, color="yellow",
                label="Ghosh-Selvan", fillrange=-9)
        else
            gs_alphas[gs_alphas .<= 1e-9] .= 1e-9 # Avoid nan values
            combined_plot = plot(gs_alphas, gs_betas, lw=1, color="yellow",
                label="Ghosh-Selvan", fillrange=-9, xscale = :log10)
        end
        plot!(marzieh_alphas, marzieh_betas, lw=1, color="green", label="Trivial
            bound", fillrange=-9)

        scatter!(alphas_omega1[betas .<= beta_max], betas[betas .<= beta_max],
            color="blue", label="\\alpha_{max} with \\Omega_1", markersize = 3)
        scatter!(alphas_omega2[betas .<= beta_max], betas[betas .<= beta_max],
            color="pink", label="\\alpha_{max} with \\Omega_2", markersize = 3)
        xlabel!(show_as_ab_b_plot ? "\\alpha\\beta" : "\\alpha")
        ylabel!("\\beta")
        yticks!(beta_min:beta_max)
        xlims!(1e-9, min(alpha_max, 1/beta_min, m/2))
        ylims!(beta_min, beta_max)
```

```julia
        title!("Frame set for Q_$m")
        display(combined_plot)
        savefig(combined_plot, "Figures/julia/gs_guaranteed_$(show_as_ab_b_plot ?
            "ab-b" :
            "a-b")_$m$(GhoshSelvanGuaranteedBound.Constants.assume_sigma_inf_conjecture
            ? "_convex" : "_non-convex").png")
    end
end
plot_guaranteed_bound(4, 100, 1, 1, 4; show_as_ab_b_plot=false,
    binary_search_for_d_with_iterations=10)
```

Listing 9.14: **scripts/gs_zz_bspline.jl**

```julia
include("../utils.jl")
include("../modules/ghosh_selvan.jl")
include("../modules/zibulski_zeevi.jl")

import .GhoshSelvan as gs
import .ZibulskiZeevi as zz

compute = true

print_progress = 1
B_symmetric = true

T1 = Float64
T2 = Double64

optim_iter_M = 20
optim_iter_s2 = 20
optim_iter = 20

alpha_max = 5
beta_max = 5
pmax = 5
qmax = 5

dalpha = 0.01
dbeta = 0.01
dw = 0.001

dt = 0.01
dv = 0.01

for m in 2:5
    if compute
        phi_hat = x -> bspline_hat(m, x)
        compact_support_hat = (-200, 200)
        compute_sums(beta, w) = gs.compute_sums_generic(phi_hat,
            compact_support_hat, beta, w)
        A_gs, B_gs, alpha_gs, beta_gs = gs.frame_bounds_grid(compute_sums,
            dalpha, dbeta, dw, alpha_max, beta_max; optim_iter_M, optim_iter_s2,
            B_symmetric, print_progress, T1, T2)

        fractions = generate_reduced_fractions_below_one(pmax, qmax)
        phi = x -> bspline(m, x)
        compact_support = (-m/2, m/2)
        is_not_frame = nothing
        A_zz, B_zz, alpha_zz, beta_zz = zz.frame_bounds_grid(phi,
            compact_support, fractions, dalpha, dbeta, dt, dv, alpha_max,
            beta_max; optim_iter, is_not_frame, print_progress, T1, T2)

        save_bounds("../Data/frame_bounds/gs_bspline_$m.npz", A_gs, B_gs,
            alpha_gs, beta_gs)
        save_bounds("../Data/frame_bounds/zz_bspline_$m.npz", A_zz, B_zz,
            alpha_zz, beta_zz)
    else
        A_gs, B_gs, alpha_gs, beta_gs =
            load_bounds("../Data/frame_bounds/gs_bspline_$m.npz")
        A_zz, B_zz, alpha_zz, beta_zz =
            load_bounds("../Data/frame_bounds/zz_bspline_$m.npz")
    end

    is_frame_gs = is_gabor_frame(A_gs, B_gs)
    is_frame_zz = is_gabor_frame(A_zz, B_zz, min_tol=1e-20)

    plt = scatter(alpha_gs[is_frame_gs], beta_gs[is_frame_gs], markersize=1 ,
        markerstrokewidth=0, label="GS is frame", alpha=0.5)
    scatter!(alpha_zz[.!is_frame_zz], beta_zz[.!is_frame_zz], markersize=1,
        markerstrokewidth=0, label="ZZ is not frame", alpha=0.5)
    scatter!(alpha_zz[is_frame_zz], beta_zz[is_frame_zz], markersize=1,
        markerstrokewidth=0, label="ZZ is frame", alpha=0.5)

    xlabel!("\\alpha")
    ylabel!("\\beta")
    display(plt)
    savefig(plt, "../Figures/julia/frame_set_bspline_$m.png")

    plt = scatter3d(alpha_gs[is_frame_gs], beta_gs[is_frame_gs],
        log10.(A_gs[is_frame_gs]), label="GS lower bond", markersize=1,
        markerstrokewidth=0, camera=(135,30), alpha=0.5)
    scatter3d!(alpha_gs[is_frame_gs], beta_gs[is_frame_gs],
        log10.(B_gs[is_frame_gs]), label="GS upper bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    scatter3d!(alpha_zz[is_frame_zz], beta_zz[is_frame_zz],
        log10.(A_zz[is_frame_zz]), label="ZZ lower bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    scatter3d!(alpha_zz[is_frame_zz], beta_zz[is_frame_zz],
        log10.(B_zz[is_frame_zz]), label="ZZ upper bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    xlabel!("\\alpha")
    ylabel!("\\beta")
    display(plt)
    savefig(plt, "../Figures/julia/frame_bounds_bspline_$m.png")
end
```

Listing 9.15: **scripts/gs_zz_hermite.jl**

```
# Run if script is executed directly
include("../utils.jl")
include("../modules/ghosh_selvan.jl")
include("../modules/zibulski_zeevi.jl")

import .ZibulskiZeevi as zz
```

```julia
import .GhoshSelvan as gs

# --- Symbolic hermite functions in Julia ---
# using Symbolics
# function hermite(n)
# @variables x
# D = Differential(x)#
# cn = (2*pi)^n * 2^(n - 1/2) * factorial(n)
# hn = (-1)^n * (cn)^(-1/2) * exp(pi * x^2) * expand_derivatives((D^n)(exp(-2 *
    pi * x^2)))
# return build_function(hn, x; expression = Val{false})
# end

hermite_2(x) = 2^(3/4)*(4*pi*x^2 - 1)*exp(-pi*x^2)/2
hermite_3(x) = ((4*pi*x^3 - 3*x)*sqrt(3)*2^(3/4)*sqrt(pi)*exp(-pi*x^2))/3
hermite_4(x) = 2^(3/4)*sqrt(3)*exp(-pi*x^2)*(16*pi^2*x^4 - 24*pi*x^2 + 3)/12
hermite_5(x) = ((16*pi^2*x^5 - 40*pi*x^3 +
    15*x)*2^(3/4)*sqrt(15)*sqrt(pi)*exp(-pi*x^2))/30

compute = true
T1 = Float64
T2 = Double64
print_progress = 2
optim_iter_M = 20
optim_iter_s2 = 20
optim_iter = 20

alpha_max = 5
beta_max = 5
dalpha = 0.01
dbeta = 0.01
dw = 0.001

compact_support = (-10, 10)
fractions = generate_reduced_fractions_below_one(5, 5)
dt = 0.01
dv = 0.01

plot_hermite_functions = true
if plot_hermite_functions
    dx = 0.01
    x = uniform_spaced_values(-3, 3, dx)

    plt = plot(x, hermite_2.(x), label="Hermite 2")
    plot!(x, hermite_3.(x), label="Hermite 3")
    plot!(x, hermite_4.(x), label="Hermite 4")
    plot!(x, hermite_5.(x), label="Hermite 5")
    title!("Hermite functions")
    xlabel!("x")
    ylabel!("y")
    display(plt)
    savefig(plt, "../Figures/julia/hermite_functions.png")
end

for (m, hn) in zip([2,3,4,5], [hermite_2, hermite_3, hermite_4, hermite_5])
    if compute
```

```julia
        phi(x) = hn.(x)
        compute_sums(beta, w) = gs.compute_sums_generic(phi, compact_support,
            beta, w)

        A_gs, B_gs, alpha_gs, beta_gs = gs.frame_bounds_grid(compute_sums,
            dalpha, dbeta, dw, alpha_max, beta_max; optim_iter_M, optim_iter_s2,
            print_progress, T1, T2)
        A_zz, B_zz, alpha_zz, beta_zz = zz.frame_bounds_grid(phi,
            compact_support, fractions, dalpha, dbeta, dt, dv, alpha_max,
            beta_max; optim_iter, print_progress, T1, T2)

        save_bounds("../Data/frame_bounds/gs_hermite_$m.npz", A_gs, B_gs,
            alpha_gs, beta_gs)
        save_bounds("../Data/frame_bounds/zz_hermite_$m.npz", A_zz, B_zz,
            alpha_zz, beta_zz)
    else
        A_gs, B_gs, alpha_gs, beta_gs =
            load_bounds("../Data/frame_bounds/gs_hermite_$m.npz")
        A_zz, B_zz, alpha_zz, beta_zz =
            load_bounds("../Data/frame_bounds/zz_hermite_$m.npz")
    end

    is_frame_gs = is_gabor_frame(A_gs, B_gs)
    is_frame_zz = is_gabor_frame(A_zz, B_zz, min_tol=1e-20)

    plt = scatter(alpha_gs[is_frame_gs], beta_gs[is_frame_gs], markersize=1,
        markerstrokewidth=0, label="GS is frame", alpha=0.5)
    scatter!(beta_gs[is_frame_gs], alpha_gs[is_frame_gs], markersize=1,
        markerstrokewidth=0, label="GS is frame", alpha=0.5)
    scatter!(alpha_zz[is_frame_zz], beta_zz[is_frame_zz], markersize=1,
        markerstrokewidth=0, label="ZZ is frame", alpha=0.5)
    scatter!(alpha_zz[.!is_frame_zz], beta_zz[.!is_frame_zz], markersize=1,
        markerstrokewidth=0, label="ZZ is not frame", alpha=0.5)
    xlabel!("\\alpha")
    ylabel!("\\beta")
    display(plt)
    savefig(plt, "../Figures/julia/frame_set_hermite_$m.png")

    plt = scatter3d(alpha_gs[is_frame_gs], beta_gs[is_frame_gs],
        log10.(A_gs[is_frame_gs]), label="GS lower bound", markersize=1,
        markerstrokewidth=0, camera=(135,30), alpha=0.5)
    scatter3d!(alpha_gs[is_frame_gs], beta_gs[is_frame_gs],
        log10.(B_gs[is_frame_gs]), label="GS upper bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    scatter3d!(beta_gs[is_frame_gs], alpha_gs[is_frame_gs],
        log10.(A_gs[is_frame_gs]), label="GS lower bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    scatter3d!(beta_gs[is_frame_gs], alpha_gs[is_frame_gs],
        log10.(B_gs[is_frame_gs]), label="GS upper bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    scatter3d!(alpha_zz[is_frame_zz], beta_zz[is_frame_zz],
        log10.(A_zz[is_frame_zz]), label="ZZ lower bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    scatter3d!(alpha_zz[is_frame_zz], beta_zz[is_frame_zz],
        log10.(B_zz[is_frame_zz]), label="ZZ upper bound", markersize=1,
        markerstrokewidth=0, alpha=0.5)
    xlabel!("\\alpha")
    ylabel!("\\beta")
    display(plt)
    savefig(plt, "../Figures/julia/frame_bounds_hermite_$m.png")
end
```

Listing 9.16: **scripts/peaks.jl**

```julia
import Pkg
Pkg.add("Plots")
Pkg.add("LinearAlgebra")
Pkg.add("Plots")
Pkg.add("DoubleFloats")
Pkg.add("Optim")
Pkg.add("NPZ")
Pkg.add("CSV")

include("../utils.jl")
include("../modules/ghosh_selvan.jl")
import .GhoshSelvan as gs
using DelimitedFiles

T1 = Float64
T2 = Double64
optim_iter_M = 10
print_progess = 0

println("ARGS:", ARGS)
m_min::Int = parse(Int64, ARGS[1])
m_max::Int = parse(Int64, ARGS[2])
beta_min::Int = parse(Int64, ARGS[3])
beta_max::Int = parse(Int64, ARGS[4])
m_range = m_min:10:m_max
beta_range = beta_min:beta_max
dbeta = 1e-2
dw = 1e-3

table = zeros(length(m_range), length(beta_range))

Threads.@threads for i in 1:length(m_range)
    m = m_range[i]
    tol = 1/(2 * m)
    d = Base.max(10^(15/m) / pi, 10)
    compact_support = (-d, d)
    phi_hat(x) = bspline_hat(m, x)
    compute_sums(beta, w) = gs.compute_sums_generic(phi_hat, compact_support,
        beta, w)

    Threads.@threads for j in 1:length(beta_range)
        local beta_int = beta_range[j]

        beta_values = uniform_spaced_values(beta_int + tol, beta_int + 1 - tol,
            dbeta; T=T1)
        alpha_beta_values = beta_values .* gs.frame_set_max_alpha(compute_sums,
            beta_values, dw; optim_iter_M, print_progess, T2)
        peak = maximum(alpha_beta_values)
        table[i,j] = peak

        println("finished m: ", m, " beta: ", beta_int)
    end
end

writedlm("../Data/tables/table_m_$m_min-$(m_max)_beta_$beta_min-$beta_max.csv", table, ',')
print(table)
```

Listing 9.17: **scripts/plot_peak_tables.jl**

```julia
include("../utils.jl")

m_min = 10; m_max = 100; beta_min = 1; beta_max = 100
m_range = m_min:10:m_max
beta_range = beta_min:beta_max

file_path = "../Data/tables/table_m_$m_min-$(m_max)_beta_$beta_min-$beta_max.csv"
data = CSV.read(file_path, CSV.Tables.matrix; header=false, delim = ',')
@assert size(data) == (length(m_range), length(beta_range))

println(size(data))

plt = plot(zscale=:log10)
xlabel!("beta")
ylabel!("peak")
# Makes a plot with beta on the x-axis and the peak values on the y-axis with colors denoting for what m
for i in 1:length(m_range)-6
    m = m_range[i]
    row = data[i, :]
    @assert size(row) == size(beta_range)
    plot!(beta_range, row, label = "m=$m")
end
display(plt)
savefig(plt, "../Figures/julia/peaks_high_m_bspline.png")

beta_range = beta_max-50:beta_max
plt = plot(zscale=:log10)
xlabel!("beta")
ylabel!("peak")
for i in 1:length(m_range)-6
    m = m_range[i]
    row = data[i, end-50:end]
    @assert size(row) == size(beta_range)
    plot!(beta_range, row, label = "m=$m")
end
display(plt)
savefig(plt, "../Figures/julia/zoooom_peaks_high_m_bspline.png")
```

Bibliography

[1] Bajwa, Waheed U., Calderbank, Robert, and Jafarpour, Sina. “Why Gabor frames? Two fundamental measures of coherence and their role in model selection”. eng. In: *Journal of Communications and Networks* 12.4 (2010), pp. 289–307. DOI: 10.1109/JCN.2010.6388466.

[2] Benedetto, John J. et al. “A frame reconstruction algorithm with applications to magnetic resonance imaging”. eng. In: *Applied and Numerical Harmonic Analysis* (2017), pp. 185–213.

[3] Berge, Eirik. “A Brief Introduction to the Feichtinger Algebra $\mathbf{S}_0(\mathbb{R})$”. und. In: (2021).

[4] Christensen, Ole. *An Introduction to Frames and Riesz Bases.* eng. Birkhäuser, 2016, 1 Online–Ressource (XXV, 704 pages).

[5] Christensen, Ole. *Differentialligninger og uendelige rækker.* dan. Polyteknisk Forlag, 2022, pp. viii, 1–355.

[6] Christensen, Ole. *Functions, Spaces, and Expansions: Mathematical Tools in Physics and Engineering.* eng. Birkhäuser Verlag, 2010, pp. XIX, 263.

[7] Christensen, Ole, Feichtinger, Hans, and Paukner, Stephan. “Gabor analysis for imaging”. eng. In: *Handbook in Imaging* (2010).

[8] Dai, Xin Rong and Sun, Qiyu. “The abc-problem for Gabor systems”. eng. In: *Memoirs of the American Mathematical Society* 244.1152 (2016), pp. 1–116. DOI: 10.1090/memo/1152.

[9] Dörfler, Monika. “Time-frequency analysis for music signals: A mathematical approach”. eng. In: *International Journal of Phytoremediation* 21.1 (2001), pp. 3–12. DOI: 10.1076/jnmr.30.1.3.7124.

[10] Feichtinger, Hans G. *Gabor Analysis and Algorithms : Theory and Applications.* eng. Ed. by Thomas Strohmer. Birkhäuser, 1998, Online–Ressource (XVI, 496 p, online resource) (unknown).

[11] Garcia, Stephan Ramon and Horn, Roger A. *A Second Course in Linear Algebra.* eng. Cambridge University Press, 2019, pp. 1–426. DOI: 10.1017/9781316218419.

[12] Ghosh, Riya and Selvan, A. Antony. “Obstructions for Gabor frames of the second order B-spline”. und. In: (2023), p. 28.

[13] Ghosh, Riya and Selvan, A. Antony. “On Gabor Frames Generated by B-splines, Totally Positive Functions, and Hermite Functions”. In: (2023).

[14] Gröchenig, Karlheinz. *Foundations of Time-Frequency Analysis.* eng. Birkhäuser, 2001, pp. 14–16.

[15] Gröchenig, Karlheinz. “Reconstruction algorithms in irregular sampling”. eng. In: *Mathematics of Computation* 59.199 (1992), pp. 181–194.

[16] Janssen, AJEM. “On rationally oversampled Weyl-Heisenberg frames”. eng. In: *Signal Processing* 47.3 (1995), pp. 239–245. DOI: 10.1016/0165-1684(95)00112-3.

[17] Lemvig, Jakob and Nielsen, Kamilla Haahr. “Counterexamples to the B-spline Conjecture for Gabor Frames”. eng. In: *Journal of Fourier Analysis and Applications* 22.6 (2016), pp. 1440–1451. DOI: 10.1007/s00041-016-9462-1.

[18] Liu, Youming. “Irregular sampling for spline wavelet subspaces”. eng. In: *Ieee Transactions on Information Theory* 42.2 (1996), pp. 623–627. DOI: 10.1109/18.485731.

[19] Lyubarskii, Yurii and Nes, Preben Gråberg. “Gabor frames with rational density”. eng. In: *Applied and Computational Harmonic Analysis* 34.3 (2013), pp. 488–494. DOI: 10.1016/j.acha.2012.09.001.

[20] Rathinaraj, Joshua David John and McKinley, Gareth H. “Gaborheometry: Applications of the discrete Gabor transform for time resolved oscillatory rheometry”. eng. In: *Journal of Rheology* 67.2 (2023), pp. 479–497. DOI: 10.1122/8.0000549.

[21] Zibulski, M and Zeevi, YY. “Oversampling in the Gabor scheme”. eng. In: *Ieee Transactions on Signal Processing* 41.8 (1993), pp. 2679–2687. DOI: 10.1109/78.229898.

Index