Approximating the Operator Norm of Schur Multipliers

A Numerical Analysis of Schur Multipliers induced by Divided Differences in Finite Dimensions

by

Nadine van Rossum

To obtain the degree of Bachelor of Science
at the Delft University of Technology.

Student number: 5575885
Project duration: April 22, 2025 – June 27, 2025
Thesis committee: Dr. M.P.T. Caspers, TU Delft, supervisor
Dr. ir. J.H. Weber, TU Delft

TU Delft

Abstract

Understanding the behavior of norms on Schur multiplier operators is of significant interest in functional analysis and applications in physics, particularly in quantum mechanics. In this study, we focus on the Schur multiplier induced by the divided difference matrix of the absolute value function $f(x) = |x|$, all set in the finite space $M_n(\mathbb{C})$. The primary objective is to approximate the operator norm $\|T_{B_f}\|$ and to analyze its behavior as a function of the Schatten norm parameter p and the matrix size n. To achieve this, various numerical methods, including brute-force sampling and gradient ascent algorithms, are explored. Among these, the Adam optimization algorithm, combining momentum and RMSProp techniques, is found to be effective in achieving accurate approximations. The results are analyzed within a theoretical framework, revealing insights into the growth of the norm as well as the effectiveness of the chosen optimization method. Future research directions are suggested, particularly in the study of multilinear Schur multipliers, which present an intriguing challenge yet to be tackled.

Laymen's summary

In quantum mechanics, we study the properties of small particles like electrons, including characteristics such as position, momentum, and energy. These properties are described using matrices and linear operators, which are mathematical tools to help us understand particle behavior. However, some characteristics, like position and momentum, cannot be measured exactly at the same time. This uncertainty is described by commutators, which show to what extent two measurements can be performed simultaneously. The more we understand commutators, the better we can describe the uncertainty in our measurements.

To study commutators, we need to calculate their mathematical bounds, which tell us the limits within which they can vary. This is where Schur multipliers come into play (see Section 3.4). Schur multipliers are a type of matrix that performs entrywise multiplication on other matrices. By establishing bounds for a certain type of Schur multiplier, we also formulate bounds for commutators.

Since calculating these bounds exactly is difficult, we use computers to approximate a certain property, the norm, of Schur multipliers. The norm gives us an idea of the size or strength of the commutator, helping us estimate its bounds, thus its behavior more easily. In this thesis, we use numerical methods to approximate these norms and improve our understanding of quantum mechanics and related fields.

Contents

1 Introduction

In many areas of functional analysis, quantum mechanics, and operator theory, understanding how norms act on operators, either in finite- or infinite-dimensional spaces, is of central importance. One particular class of operators, known as Schur multipliers, is defined via entrywise multiplication and has proven to be widely applicable in those areas. These operators are linear and admit operator norms whose behavior is of significant interest, as the norm measures how the operator amplifies inputs and indicates the stability and boundedness of the transformation.

A certain type of Schur multiplier appears in quantum mechanics, as will be the focus of this work. Additionally, Schur multipliers in general are relevant in multiple other areas. They are essential in perturbation theory, where they are used to study the stability and behavior of operators under small changes. They facilitate the analysis of how perturbations affect operator norms, as well as spectral properties. In harmonic analysis, Schur multipliers are used to understand the boundedness and behavior of convolution operators. They play a crucial role in the study of Fourier multipliers and their applications to various function spaces. Thus, in addition to their relevance in quantum mechanics, Schur multipliers are of interest in several other fields [11].

This thesis focuses on analyzing and approximating the operator norm of a specific type of Schur multiplier, namely one induced by the divided difference of a given function, in a finite-dimensional setting. More precisely, we consider the operator $T_{B_f} : M_n(\mathbb{C}) \to M_n(\mathbb{C})$ defined by $T_{B_f}(B) = B_f * B$, where B_f is a matrix composed of divided differences of a function f over values $\lambda_1, \ldots, \lambda_n$, and $*$ denotes the Schur (entrywise) product.
Divided differences arise naturally in the study of commutators of operators in quantum mechanics, where noncommutativity reflects fundamental uncertainty principles. In this context, the function $f(x) = |x|$ plays a central role, which is a non-differentiable but Lipschitz continuous function. As such, the operator norm $\|T_{B_f}\|$ for this f contains meaningful information about how functions of noncommuting quantum observables behave. This norm is defined as follows:

$$\|T_{B_f}\| = \sup_{\substack{B \in M_n(\mathbb{C}) \\ \|B\|_p \neq 0}} \frac{\|B_f * B\|_p}{\|B\|_p}. \tag{1.1}$$

Although this norm can be difficult to compute exactly, especially as n grows, it can be bounded theoretically and approximated numerically. Previous research has established several upper and lower bounds, revealing asymptotic behaviors in both the parameter p of the Schatten norm and the matrix size n. These theoretical insights provide guidance, but do not always give tight estimates in practical settings. Consequently, this work aims to approximate the supremum given in Equation 1.1 numerically, with particular attention to the behavior of the norm as a function of both p and n.

Therefore, several numerical methods are explored, including brute-force sampling, structured matrices $A(\ell)$, and gradient ascent algorithms. These last algorithms are widely used in machine learning, mostly due to

their simplicity, versatility and efficiency. A gradient ascent algorithm finds a maximum of a function by iteratively computing the gradient at the current point in the function and then progressing into the direction of steepest ascent. The method can be applied to a a wide range of optimization problems. By only requiring the gradient and being computationally cheap, it handles both small and large-scale problems effectively.

Among the multiple types of gradient ascent, the Adam optimization algorithm is found to be particularly effective in dealing with the non-convexity and instability of the norm function. This algorithm combines two optimization techniques, momentum and RMSProp. Hence, Adam optimization is implemented in Python to obtain an approximation for the norm function. This norm function is then computed as a function of both the parameter p and the matrix size n. Using a theoretical background, these results are analyzed with the goal of evaluating the effectiveness of the chosen optimization method in this context.

The structure of this thesis is as follows. Chapter 2 introduces the necessary preliminaries from functional analysis and operator theory. Chapter 3 discusses the Schatten norms and defines the Schur multiplier operator. In addition, the use of the divided difference matrix is motivated and several prior results concerning this subject are presented. Chapter 4 presents the numerical methods used to approximate the operator norm. Chapter 5 contains the resulting visualizations and an analysis of the effectiveness of Adam optimization. The thesis concludes with a conclusion, a discussion of the findings, and suggestions for future research.

2

Preliminaries

An introduction is given on functional analysis and operator theory so that one may study the Schatten p-norm and Schur multipliers. The given content is based on [1], [12], and [14]. The reader is assumed to be familiar with basic linear algebra, such as orthogonal bases and the rank of a matrix. Some basic real analysis is assumed to be known as well, such as the notions of completeness and separability.

Notation

The following notation is commonly used in this thesis.

- $M_{n\times m}(\mathbb{C})$ is the set of $n \times m$ matrices with entries in $\mathbb{C}$. If $n = m$ it is shortened to $M_n(\mathbb{C})$.
- A matrix $A \in M_n(\mathbb{C})$ is indicated with a capital letter. For an entry at position i, j we write $a_{i,j}$.
- The adjoint of a matrix A, $\overline{A}^{\mathsf{T}}$, is denoted by A^*.
- For a linear operator $T : X \to Y$, we may write $T(x) = Tx$ for $x \in X$.

Throughout this chapter, let X, Y be normed vector spaces and assume a vector space is taken over $M_n(\mathbb{C})$, unless stated otherwise.

Definition 2.1. A linear operator $T : X \to Y$ is bouned if there exists a constant $C \geq 0$ such that

$$\|Tx\| \leq C\|x\|, \quad \text{for all } x \in X.$$

Such an operator is then called a bounded operator. The space of all such operators is denoted by $B(X, Y)$, where we write $B(X)$ if $X = Y$.

Definition 2.2. Let $T \in B(X, Y)$. Then the operator norm of T is given by

$$\|T\|_{op} = \inf\{C \geq 0 : \text{for all } x \in X : \|Tx\| \leq C\|x\|\}$$

In case it is clear from context, we denote the operator norm of T as $\|T\|$.

Theorem 2.3. Let $T \in B(X, Y)$. If $X \neq 0$, then $\|T\| = \sup_{\|x\|\leq 1} \|Tx\|$.

Proof. Let $C \geq 0$ be such that for all $x \in X$, we have $\|Tx\| \leq C\|x\|$. Then, when considering the supremum over all x with $\|x\| \leq 1$, we see

$$\sup_{\|x\|\leq 1} \|Tx\| \leq C.$$

Since this holds for all admissible constants C, it follows that

$$\sup_{\|x\|\leq 1} \|Tx\| \leq \|T\|.$$

The opposite inequality follows from the fact that for all $x \in X\backslash\{0\}$, we have:

$$\|Tx\| = \|\frac{1}{\|x\|}Tx\|\|x\| = \|T\frac{x}{\|x\|}\|\|x\| \leq \sup_{\|x\|\leq 1}\|Tx\|\|x\|.$$

Thus $\sup_{\|x\|\leq 1}\|Tx\|$ is an admissible constant. Hence, we conclude

$$\|T\| = \sup_{\|x\|\leq 1}\|Tx\|.$$

□

Remark 2.4. Note that for $X \neq 0$, $\|T\| = \sup_{x\neq 0}\frac{\|Tx\|}{\|x\|} = \sup_{\|x\|=1}\|Tx\|$.

There is a convenient relationship between boundedness and continuity for linear operators, which is given in Theorem 2.5. The proof is adapted from and based on [14, Theorem 2.5].

Theorem 2.5. Let T be a linear operator. Then

$$T \text{ is bounded} \iff T \text{ is continuous}$$

Proof. For $\|T\| = 0$ the proof is trivial, thus assume $\|T\| \neq 0$.
$\Rightarrow$ Suppose T is bounded. Then we have

$$\|Tx - Tx'\| = \|T(x - x')\| \leq \|T\|\|x - x'\|$$

Hence T is Lipschitz continuous and therefore continuous.
$\Leftarrow$ Suppose T is continuous. In particular, we thus have that T is continuous in 0. Then, by definition, for every $\epsilon > 0$ there exists a $\delta > 0$ such that if $\|x\| < \delta$, then $\|Tx\| < \epsilon$. Set $\epsilon = 1$ and $M = \frac{2}{\delta}$. Now let x be arbitrary in X. If $x = 0$, we have that $Tx = 0$ (by linearity) and thus $\|Tx\| = 0 \leq M\|x\|$. Suppose $x \neq 0$. We consider $y = \frac{x}{M\|x\|}$. Since $\|y\| < \delta$, by continuity we have that $\|Ty\| < 1$. Then using the linearity of T we find:

$$\|Ty\| < 1 \iff \|T\frac{x}{M\|x\|}\| < 1 \iff \|Tx\| < M\|x\|.$$

Thus, T is bounded. This proves the if and only if statement. □

Throughout this thesis we will mostly be working in the space $M_n(\mathbb{C})$. The terms matrix and linear operator can in that case be used interchangeably. However, for some definitions we will consider linear operators in different spaces. These are introduced below.

Definition 2.6. A Banach space is a normed complete vector space.

Definition 2.7. A Hilbert space is a complete inner product space.

Since every inner product space can be made into a normed space [12], every Hilbert space is a Banach space. Thus Hilbert spaces are a special case of Banach spaces.
Another notion that should be introduced is that of compact operators. The reader might recall that a topological space is said to be compact if it is both complete and totally bounded [1].

Definition 2.8. Let X, Y be Banach spaces and U the closed unit ball in X. An operator $T : X \to Y$ is said to be compact if the closure of $T(U)$ is compact in Y.

Remark 2.9. Note that every linear operator $T : M_n(\mathbb{C}) \to M_n(\mathbb{C})$ is compact, since $M_n(\mathbb{C})$ is finite-dimensional.

We will show that a compact operator is automatically bounded. Note that compact subsets are necessarily bounded. Since the closure of the image of the open unit ball, $\overline{T(U)}$, is compact in Y, it follows that there exists a constant $M > 0$ such that

$$\|y\|_Y \leq M \quad \text{for all } y \in \overline{T(U)}.$$

Because $T(U) \subseteq \overline{T(U)}$, this implies that

$$\|Tx\|_Y \leq M \quad \text{for all } x \in U,$$

showing that T maps the unit ball U into a bounded set in Y.
To extend this boundedness to all of X, consider an arbitrary vector $x \in X$. If $x = 0$, the result trivially holds. For $x \neq 0$, define

$$u := \frac{x}{\|x\|_X},$$

which lies in the closed unit ball U. By linearity of T, we have

$$Tx = T(\|x\|_X u) = \|x\|_X Tu,$$

and therefore

$$\|Tx\|_Y = \|x\|_X \|Tu\|_Y \leq \|x\|_X M.$$

This inequality shows that T is bounded on all of X, completing the proof. Thus we may consider the operator norm on compact operators. This result will be used later on in this thesis. In general, we will denote by $K(X, Y)$ the compact operators from X to Y, where we again write $K(X)$ if $X = Y$.

A few last definitions are given for matrices in $M_n(\mathbb{C})$.

Definition 2.10. Let $B \in M_n(\mathbb{C})$. We say that B is self-adjoint if

$$\langle Bx, y\rangle = \langle x, By\rangle \quad \text{for all } x, y \in \mathbb{C}^n.$$

Or equivalently, $B = B^*$.

Definition 2.11. Let $B \in M_n(\mathbb{C})$ be a diagonalizable matrix. Then B admits an eigendecomposition, meaning there exists an invertible matrix $V \in M_n(\mathbb{C})$ and a diagonal matrix $\Lambda \in M_n(\mathbb{C})$ such that

$$B = V\Lambda V^{-1},$$

where the columns of V are the eigenvectors of B, and the diagonal entries of Λ are the corresponding eigenvalues. If B is self-adjoint, we can take V unitary.

Definition 2.12. Let $A \in M_n(\mathbb{C})$ and consider A^*A. Then we can write

$$\langle A^*Ax, y\rangle = \langle Ax, Ay\rangle = \langle x, A^*Ay\rangle,$$

thus we see that A^*A is self-adjoint. Hence, it can be written as $A^*A = UDU^*$, for U a unitary matrix and D a diagonal matrix with the eigenvalues of A^*A on the diagonal. We now define:

$$|A| := \sqrt{A^*A} := UD^{1/2}U^*, \quad \text{where} \quad (d^{1/2})_{i,i} = \sqrt{d_{i,i}}.$$

We write here $\sqrt{A^*A}$ since $|A|^2 = UDU^* = A^*A$. We must check if we can indeed take the square root of the diagonal elements of D, i.e. the eigenvalues of A^*A. Let λ be an eigenvalue of A^*A and x a corresponding eigenvector. Below is shown that λ is indeed non-negative:

$$\lambda\|x\|^2 = \langle \lambda x, x\rangle = \langle A^*Ax, x\rangle = \langle Ax, Ax\rangle = \|Ax\|^2.$$

After dividing by $\|x\|^2$ we see the desired result.

3

The Schatten p-norm on linear operators and Schur multipliers

3.1. Singular values and the norm

In this section, we introduce the Schatten p-norm on matrices $A \in M_n(\mathbb{C})$, which we will define in terms of singular values. To this end, we will start by defining singular values. For the sake of completeness we will do this on a Hilbert space H. After introducing the p-norm, we shall return to the finite matrix space $M_n(\mathbb{C})$.

Definition 3.1. Let H, G be Hilbert spaces, and $T : H \to G$ a compact linear operator. We then define the singular values of T for $n \in \mathbb{N}$ as follows:

$$s_n(T) := \inf\{\|T - S\| \mid S \in K(H,G) \text{ and } \operatorname{rank}(S) < n\}$$

If $n = 1$, then $s_1(T) = \|T\|$, and if $n > \operatorname{rank}(T)$, then $s_n(T) = 0$. Also note that $(s_n(T))_{n=1}^{\infty}$ is a non-negative decreasing sequence, since:

$$\{\|T - S\| \mid \operatorname{rank}(S) < n\} \subset \{\|T - S\| \mid \operatorname{rank}(S) < n+1\}.$$

Remark 3.2. In [14] it is shown that the singular values of A coincide with the eigenvalues of $|A|$, which is used to compute the approximations in Chapter 5.

These singular values help define subspaces of the linear operators that are compact on H, namely the Schatten classes.

Definition 3.3. Let H be a separable Hilbert space. For $p \in [1,\infty)$, we define the Schatten class $S_p(H)$ as follows:

$$S_p(H) := \{T \in K(H) \mid \sum_{k=1}^{\infty} s_k(T)^p < \infty\}.$$

For $p = \infty$, we define $S_\infty(H) := K(H)$.

Definition 3.4. Let $p \in [1,\infty]$ and H a Hilbert space. For any $T \in S_p(H)$ and $p < \infty$, we define the Schatten p-norm as:

$$\|T\|_p := \left(\sum_{k=1}^{\infty} s_k(T)^p\right)^{1/p}.$$

For $p = \infty$, the Schatten ∞-norm is given by:

$$\|T\|_\infty = \sup_k s_k(T) = s_1(T).$$

This norm is also known as the Spectral Norm.

When working in a Hilbert space H, the Schatten p-norm will only be finite if the operator on which it acts is in the Schatten class. Thus these Schatten classes define the domain of the Schatten p-norm. However, when working in a finite space such as $M_n(\mathbb{C})$, the norm is always finite. Thus when defining the Schatten p-norm then, the Schatten classes are not needed, as shown below:

Definition 3.5. Let $p \in [1,\infty]$. For any $A \in M_n(\mathbb{C})$ and $p < \infty$, we define the Schatten p-norm as:

$$\|A\|_p := \left(\sum_{k=1}^{n} s_k(A)^p\right)^{1/p}.$$

For $p = \infty$, the Schatten ∞-norm is given by:

$$\|A\|_\infty = \sup_k s_k(A) = s_1(A).$$

Remark 3.6. We can regard this norm as the ℓ^p norm of $(s_k(A))_{k=1}^n$. Then we see that for $q \geq p$, we have $\|A\|_q \leq \|A\|_p$.

In combination with this norm, $(M_n(\mathbb{C}), \|.\|_p)$ is a Banach space. We will show this by proving that $\|.\|_p$ is in fact a norm on $M_n(\mathbb{C})$, since we know that every finite-dimensional normed space is a Banach space [12].

3.1.1. Proof of Norm Properties

In this section, let $A \in M_n(\mathbb{C})$.

i $0 \leq \|A\|_p < \infty$.

The singular values of A are all non-negative, so the norm is always ≥ 0. Since we are considering finite matrices, the sum of the singular values of these matrices are finite as well.

ii $\|A\|_p = 0 \iff A = 0$ (the zero matrix).

We see that this property holds as follows:

$$\begin{aligned} \|A\|_p = 0 &\iff s_i(A) = 0 \quad \forall 1 \leq i \leq n \\ &\iff A = 0. \end{aligned}$$

iii $\|\alpha A\|_p = |\alpha| \|A\|_p \quad , \alpha \in \mathbb{C}$.

Let $\alpha \in \mathbb{C}$. It is easily verified that $\forall i \in \{1,\ldots,n\}$, $s_i(\alpha A) = |\alpha| s_i(A)$. Hence for the p-norm we obtain the following:

$$\begin{aligned} \|\alpha A\|_p &= \left(\sum_{i=1}^{n} s_i(\alpha A)^p\right)^{1/p} \\ &= \left(\sum_{i=1}^{n} (|\alpha| s_i(A))^p\right)^{1/p} \\ &= \left(|\alpha|^p \sum_{i=1}^{n} s_i(A)^p\right)^{1/p} \\ &= |\alpha| \left(\sum_{i=1}^{n} s_i(A)^p\right)^{1/p} \\ &= |\alpha| \|A\|_p. \end{aligned}$$

Lastly, we want to show that the triangle inequality holds for this norm, that is, for $T, S \in M_n(\mathbb{C})$, we want to show $\|T+S\|_p \leq \|T\|_p + \|S\|_p$. It turns out that this is quite a difficult proof and we will need to introduce several lemmas before being able to prove this identity. These lemmas are based on similar lemmas in [6], but adapted for the specific case of the vector space $M_n(\mathbb{C})$.We will start by introducing a decomposition of an $n \times n$ matrix T that is based on its singular values.

Lemma 3.7. If T is an $n \times n$ matrix in $M_n(\mathbb{C})$, then we can write it as

$$T = \sum_{j=1}^{n} s_j(T)\langle \cdot, e_j\rangle f_j,$$

for certain orthonormal sequences $(e_j)_{j=1}^n$ and $(f_j)_{j=1}^n$ in $\mathbb{C}^n$. Here, $s_j(T)$ are the singular values of T, hence we call this the Singular Value Decomposition.

Proof. Let T be an $n \times n$ matrix in $M_n(\mathbb{C})$. Then we can obtain a Singular Value Decomposition $T = UDV^*$, with U, V unitary and D a diagonal matrix. We then write

$$D = \begin{pmatrix} s_1 & 0 & \dots & 0 \\ 0 & s_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & s_n \end{pmatrix},$$

with $s_1 \geq s_2 \geq \cdots \geq s_n$. Note that we can obtain this D by swapping rows and/or columns of U, V. Thus $s_j, 1 \leq j \leq n$, are the singular values of T.
We now choose an orthonormal basis of $\mathbb{C}^n$, say $h_1, h_2, \dots, h_n$, which are the eigenvectors of D, i.e.:

$$h_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, h_2 = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \dots, h_n = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

Take $e_j = V^* h_j, f_j = U h_j$. Since U and V are unitary, $(e_j)_{j=1}^n$ and $(f_j)_{j=1}^n$ are orthonormal sequences.
Then for a vector $w \in \mathbb{C}^n$ we see that

$$\begin{aligned} Tw &= UDVw \\ &= U \sum_{j=1}^{n} s_j(T)\langle Vw, h_j\rangle h_j \\ &= \sum_{j=1}^{n} s_j(T)\langle Vw, h_j\rangle U h_j \\ &= \sum_{j=1}^{n} s_j(T)\langle w, V^* h_j\rangle U h_j \\ &= \sum_{j=1}^{n} s_j(T)\langle w, e_j\rangle f_j. \end{aligned}$$

□

Remark 3.8. It should be noted that a different definition for singular values is used in the proof above than what was given in Definition 3.1. These definitions are equal, but the proof of this is beyond the scope of this thesis.

Next, we will use this decomposition to give an expression for the sum of the singular values that will be useful in the lemma that follows.

Lemma 3.9. Let $T \in M_n(\mathbb{C})$. Then for $m \in \mathbb{N}$,

$$\sum_{k=1}^{m} s_k(T) = \max \sum_{k=1}^{m} |\langle Tg_k, h_k\rangle| = \max \sum_{k=1}^{m} |\langle Tg_k, h_k\rangle|,$$

where we take the maximum over all orthonormal sequences $(g_k)_{k=1}^m$ and $(h_k)_{k=1}^m$ in $\mathbb{C}^n$.

Proof. By Lemma 3.7, we can write $T = \sum_{j=1}^{n} s_j(T)\langle \cdot, e_j\rangle f_j$. Thus by taking $(e_k)_{k=1}^m$ and $(f_k)_{k=1}^m$, we obtain:

$$\begin{aligned}\sum_{k=1}^{m}\langle Te_k, f_k\rangle &= \sum_{k=1}^{m}\langle \sum_{j=1}^{n} s_j(T)\langle e_k, e_j\rangle f_j, f_k\rangle \\ &= \sum_{k=1}^{m}\langle s_k(T) f_k, f_k\rangle \\ &= \sum_{k=1}^{m} s_k(T).\end{aligned}$$

Now we take the absolute value and the supremum over all orthonormal sequences to obtain:

$$\sum_{k=1}^{m} s_k(T) \le \sup|\sum_{k=1}^{m}\langle Tg_k, h_k\rangle| \le \sup\sum_{k=1}^{m}|\langle Tg_k, h_k\rangle|,$$

where we applied the triangle inequality of the absolute value for the second inequality. We now show the other inequality. We start by substituting the SVD of T in $\sum_{k=1}^{m}|\langle Tg_k, h_k\rangle|$:

$$\begin{aligned}\sum_{k=1}^{m}|\langle Tg_k, h_k\rangle| &= \sum_{k=1}^{m}|\langle\sum_{j=1}^{n} s_j(T)\langle g_k, e_j\rangle f_j, h_k\rangle| \\ &= \sum_{k=1}^{m}|\sum_{j=1}^{n} s_j(T)\langle g_k, e_j\rangle\langle f_j, h_k\rangle| \\ &\le \sum_{j=1}^{n} s_j(T)\sum_{k=1}^{m}|\langle g_k, e_j\rangle\langle f_j, h_k\rangle|\end{aligned} \tag{3.1}$$

First, consider $\sum_{k=1}^{m}|\langle g_k, e_j\rangle\langle f_j, h_k\rangle|$. We consider this the standard inner product of $(|\langle g_1, e_j\rangle|, |\langle g_2, e_j\rangle|, \ldots, |\langle g_n, e_j\rangle|)$ and $(|\langle f_j, h_1\rangle|, |\langle f_j, h_2\rangle|, \ldots, |\langle f_j, h_n\rangle|)$. Then we can apply the Cauchy-Schwarz inequality to obtain:

$$\begin{aligned}\sum_{k=1}^{m}|\langle g_k, e_j\rangle\langle f_j, h_k\rangle| &\le (\sum_{k=1}^{m}|\langle g_k, e_j\rangle|^2)^{\frac{1}{2}}(\sum_{k=1}^{m}|\langle f_j, h_k\rangle|^2)^{\frac{1}{2}} \\ &\le \|e_j\|\|f_j\| = 1.\end{aligned}$$

We perform similar steps after switching the order of summation to find:

$$\begin{aligned}\sum_{j=1}^{n}\sum_{k=1}^{m}|\langle g_k, e_j\rangle\langle f_j, h_k\rangle| &= \sum_{k=1}^{m}\sum_{j=1}^{n}|\langle g_k, e_j\rangle\langle f_j, h_k\rangle| \\ &\le \sum_{k=1}^{m}[(\sum_{j=1}^{n}|\langle g_k, e_j\rangle|^2)^{\frac{1}{2}}(\sum_{j=1}^{n}|\langle f_j, h_k\rangle|^2)^{\frac{1}{2}}] \\ &\le \sum_{k=1}^{m}\|g_k\|\|h_k\| = \sum_{k=1}^{m} 1 = m.\end{aligned}$$

For this next part, we define $c_j := \sum_{k=1}^{m}|\langle Tg_k, e_j\rangle\langle f_k, h_j\rangle|$. Thus, we have shown that $0 \le c_j \le 1$ and $\sum_{j=1}^{m} c_j \le m$. Recalling $s_{n+1}(T) = 0$, we now rewrite:

$$\sum_{j=1}^{n} s_j(T)c_j = \sum_{j=1}^{n}\sum_{k=j}^{n}(s_k(T) - s_{k+1}(T))\,c_j = \sum_{k=1}^{n}\sum_{j=1}^{k}(s_k(T) - s_{k+1}(T))\,c_j$$

Thus, we find:

$$\sum_{j=1}^{n} s_j(T)c_j = \sum_{k=1}^{n}(s_k(T) - s_{k+1}(T))\sum_{j=1}^{k} c_j.$$

Since $c_j \geq 0$, we find $\sum_{j=1}^{k} c_j \leq \sum_{j=1}^{n} c_j \leq m$. Combined with the earlier result that $c_j \leq 1$, we find that $\sum_{j=1}^{k} c_j \leq \sum_{j=1}^{min\{k,m\}} 1$. Now since $s_k(T) - s_{k+1}(T) \geq 0$, we find

$$\begin{aligned}\sum_{j=1}^{n} s_j(T)c_j &= \sum_{k=1}^{n}\sum_{j=1}^{k} (s_k(T) - s_{k+1}(T))\, c_j \\ &\leq \sum_{k=1}^{n} (s_k(T) - s_{k+1}(T)) \sum_{j=1}^{min\{k,m\}} 1 \\ &= \sum_{j=1}^{m}\sum_{k=j}^{n} (s_k(T) - s_{k+1}(T)) \\ &= \sum_{j=1}^{m} s_j(T)\end{aligned}$$

Now, taking the supremum over all orthonormal sequences, this inequality, together with the result shown in Equation 3.1, indeed gives:

$$\sup \sum_{k=1}^{m} \left|\langle Tg_k, h_k\rangle\right| \leq \sum_{k=1}^{m} s_k(T),$$

thus proving equality. Lastly, we see that it is indeed a maximum by choosing $(e_k)_{k=1}^{m}$ and $(f_k)_{k=1}^{m}$ as the sequences in the inproduct. □

Lemma 3.10. For $T, S \in M_n(\mathbb{C})$ and $M \in \mathbb{N}$ the following holds:

$$\sum_{k=1}^{m} s_k(T+S) \leq \sum_{k=1}^{m} s_k(T) + \sum_{k=1}^{m} s_k(S)$$

Proof. We use Lemma 3.9 to obtain the following:

$$\sum_{k=1}^{m} s_k(T+S) = \max \sum_{k=1}^{m} |\langle (T+S)g_j, h_j\rangle|.$$

Since we know the triangle inequaltiy holds for $|\cdot|$, we can apply it here:

$$\max \sum_{k=1}^{m} |\langle (T+S)g_j, h_j\rangle| \leq \max \sum_{k=1}^{m} |\langle (Tg_k, h_k\rangle| + \max \sum_{k=1}^{m} |\langle (Sg_k, h_k\rangle|.$$

After applying Lemma 3.9 once more, we obtain the result we wanted to prove. □

Remark 3.11. We have proven Lemma 3.9 and 3.10 for general $m \in \mathbb{N}$. However, since in this thesis we only consider finite square matrices, it would have sufficed to show these results for $m = n$, where n is the dimension of the matrices.

Before proving the triangle inequality at last, a final Lemma is introduced.

Lemma 3.12. Let $f : \mathbb{R} \to \mathbb{R}$ be a convex increasing function, $n \in \mathbb{N}$, and $(a_i)_{i=1}^{n}$, $(b_i)_{i=1}^{n}$ be two decreasing sequences. Define the partial sums $A_k := \sum_{i=1}^{k} a_i$ and $B_k := \sum_{i=1}^{k} b_i$. If for all $k \in \{1, \ldots, n\}$ we have $A_k \leq B_k$, then:

$$\sum_{i=1}^{n} f(a_i) \leq \sum_{i=1}^{n} f(b_i).$$

Proof. We aim to show that:

$$\sum_{i=1}^{n} \big(f(b_i) - f(a_i)\big) \geq 0.$$

Suppose for some (possibly several) j we have $a_j = b_j$. Then we can remove these a_j and b_j from the sequence and still end up with the same inequality. Therefore, without loss of generality, we assume that $a_i \neq b_i$ for all $i \in \{1, \ldots, n\}$.

We may then define $c_i = \frac{f(b_i) - f(a_i)}{b_i - a_i}$, which allows us to rewrite the sum as:

$$\sum_{i=1}^{n} \big(f(b_i) - f(a_i)\big) = \sum_{i=1}^{n} c_i(b_i - a_i).$$

Since f is increasing, we see that:

- If $b_i > a_i$, then $f(b_i) \geq f(a_i)$, implying $c_i \geq 0$;
- If $b_i < a_i$, then $f(b_i) \leq f(a_i)$, implying $c_i \geq 0$.

Thus in both cases we have $c_i \geq 0$.

We now claim that $(c_i)_{i=1}^n$ is a decreasing sequence. Define $R(x,y) := \frac{f(x)-f(y)}{x-y}$ for $x \neq y$. Note: $c_i = R(b_i, a_i)$. We have that f is convex if and only if R is increasing in both variables. In the case where $a_i, a_{i+1}, b_i, b_{i+1}$ are all distinct, we find that $c_i = R(a_i, b_i) \geq R(a_{i+1}, b_{i+1}) = c_{i+1}$. In the case that two terms are equal, the inequality follows from a single step and the symmetry of R in its arguments.
Next, using $a_i = A_i - A_{i-1}$ and $b_i = B_i - B_{i-1}$, we obtain:

$$\sum_{i=1}^{n} \big(f(b_i) - f(a_i)\big) = \sum_{i=1}^{n} c_i b_i - c_i a_i = \sum_{i=1}^{n} c_i(B_i - B_{i-1}) - \sum_{i=1}^{n} c_i(A_i - A_{i-1}) = \sum_{i=1}^{n} c_i(B_i - A_i) - \sum_{i=1}^{n} c_i(B_{i-1} - A_{i-1}).$$

We then use the fact that $A_0 = B_0 = 0$ to obtain:

$$\sum_{i=1}^{n} c_i(B_{i-1} - A_{i-1}) = \sum_{i=0}^{n-1} c_{i+1}(B_i - A_i) = \sum_{i=1}^{n-1} c_{i+1}(B_i - A_i)$$

Substituting this in the equality above gives:

$$\sum_{i=1}^{n} \big(f(b_i) - f(a_i)\big) = c_n(Bn - A_n) + \sum_{i=1}^{n-1} (c_i - c_{i+1})(B_i - A_i).$$

Since c_i is a decreasing sequence, we have $c_i - c_{i+1} \geq 0$, and since $B_i - A_i \geq 0$ by assumption, all terms in the sum are non-negative. Therefore, we conclude that:

$$\sum_{i=1}^{n} \big(f(b_i) - f(a_i)\big) \geq 0.$$

Thus, we have proven that:

$$\sum_{i=1}^{n} f(a_i) \leq \sum_{i=1}^{n} f(b_i),$$

as desired. □

We are now able to prove the triangle inequality for the Schatten p-norm. First, we use Lemma 3.10 to obtain:

$$\sum_{k=1}^{n} s_k(T+S) \leq \sum_{k=1}^{n} s_k(T) + \sum_{k=1}^{n} s_k(S) = \sum_{k=1}^{n} (s_k(T) + s_k(S)).$$

Now consider $f(x) = x^p$ for $p \geq 1$. Note that $f'(x) = px^{p-1}$ and $f''(x) = p(p-1)x^{p-2}$. Since the sums that we apply this function to are all positive, we see that both $f'(x) \geq 0$ and $f''(x) \geq 0$. Hence f is both convex and increasing. Thus we may apply Lemma 3.12 to obtain:

$$\sum_{k=1}^{n} (s_k(T+S))^p \leq \sum_{k=1}^{n} (s_k(T) + s_k(S))^p,$$

where we took $a_k = s_k(T+S)$ and $b_k = s_k(T) + s_k(S)$. Then, since $g(x) = x^{\frac{1}{p}}$ is increasing, we can apply it to both sides and maintain the inequality. That is, we obtain:

$$\Big(\sum_{k=1}^{n} (s_k(T+S))^p\Big)^{\frac{1}{p}} \leq \Big(\sum_{k=1}^{n} (s_k(T) + s_k(S))^p\Big)^{\frac{1}{p}}.$$

Since the right-hand side is the p-norm for the vector $(s_1(T) + s_1(S), s_2(T) + s_2(S), \ldots, s_n(T) + s_n(S))$ in $\mathbb{C}^n$, for which we know the triangle inequality to be true, we obtain the desired result:

$$\|T+S\|_p = \Big(\sum_{k=1}^{n} (s_k(T+S))^p\Big)^{\frac{1}{p}} \leq \Big(\sum_{k=1}^{n} (s_k(T) + s_k(S))^p\Big)^{\frac{1}{p}} \leq \Big(\sum_{k=1}^{n} s_k(T)^p\Big)^{\frac{1}{p}} + \Big(\sum_{k=1}^{n} s_k(S)^p\Big)^{\frac{1}{p}} = \|T\|_p + \|S\|_p.$$

Hence we have shown that $(M_n(\mathbb{C}), \|\cdot\|_p)$ is a Banach space. In the next section an operator on elements in this space is introduced.

3.2. Schur Multiplier

In order to define an operator on matrices in $M_n(\mathbb{C})$, we will start by giving the definition of a Schur product. This can be described quite simply as the entrywise multiplication of two matrices.

Definition 3.13. Let $A, B \in M_n(\mathbb{C})$. We define the Schur product of A and B as follows:

$$A * B = (a_{ij} b_{ij})_{i,j=1}^{n}$$

Thus, $A * B$ is also an $n \times n$ matrix and is the result of entrywise multiplication of A and B.

It is easily verified that the Schur product satisfies the following properties:

- Commutativity: $A * B = B * A$.
- Associativity: $A * (B * C) = (A * B) * C$.
- Distributivity over addition: $A * (B + C) = A * B + A * C$.
- Compatibility with scalar multiplication:

  $$\alpha(A * B) = (\alpha A) * B = A * (\alpha B),$$

  for any scalar α.

With this form of multiplication, we can define a linear operator: the Schur multiplier.

Definition 3.14. Let $A \in M_n(\mathbb{C})$. Then we can define a map from $M_n(\mathbb{C})$ onto itself, $T_A : M_n(\mathbb{C}) \to M_n(\mathbb{C})$, by $M_A(B) \coloneqq A * B$ for $B \in M_n(\mathbb{C})$. Or, equivalently, $M_A((b_{ij})_{i,j=1}^{n}) \coloneqq (a_{ij} b_{ij})_{i,j=1}^{n}$. We call this map a Schur multiplier.

The Schur multiplier T_A is a linear operator. To see this, let $B, C \in M_n(\mathbb{C})$ and scalars $\alpha, \beta \in \mathbb{C}$:

$$\begin{aligned} T_A(\alpha B + \beta C) &= A * (\alpha B + \beta C) \\ &= \left(a_{ij}\right) * \left(\alpha b_{ij} + \beta c_{ij}\right) \\ &= \left(\alpha a_{ij} b_{ij} + \beta a_{ij} c_{ij}\right) \\ &= \alpha(A * B) + \beta(A * C) \\ &= \alpha T_A(B) + \beta T_A(C). \end{aligned}$$

Before being able to consider the operator norm on T_A, we must first show that it is bounded.

On a finite-dimensional space, all norms are equivalent. Thus we only have to show boundedness for a single norm. To this end, let $B \in M_n(\mathbb{C})$. We consider the following norm:

$$\|B\| = \max_j \sum_{i=1}^{n} |b_{ij}|$$

This norm is known as the 1-norm of a matrix. Applying T_A, we find:

$$\begin{aligned} \|T(B)\| &= \|A * B\| \\ &= \max_j \sum_{i=1}^{n} |a_{ij} b_{ij}| \\ &= \max_j \sum_{i=1}^{n} |a_{ij}||b_{ij}| \\ &\leq \max_{i,j} |a_{ij}| \cdot \max_j \sum_{i=1}^{n} |b_{ij}| \\ &= M\|B\|, \end{aligned}$$

where we define $M = \max_{i,j} |a_{ij}|$, i.e., the entry of A with the highest absolute value. Thus T_A is bounded.

3.3. Operator norm on a Schur multiplier

In the previous section, it has been shown that the Schur multiplier T_A is bounded, thus we may consider the operator norm. This gives us the following:

$$\|T_A\| = \sup_{\substack{B\in M_n(\mathbb{C}) \\ \|B\|\neq 0}} \frac{\|T_A(B)\|_p}{\|B\|_p} = \sup_{\substack{B\in M_n(\mathbb{C}) \\ \|B\|\neq 0}} \frac{\|A*B\|}{\|B\|}. \tag{3.2}$$

In this thesis, we aim to approximate the norm on a specific Schur multiplier and analyse its behaviour. This Schur multiplier is the operator that performs Schur Multiplication with a divided difference matrix. The definition of this matrix is given below.

Definition 3.15. Let $f:\mathbb{R}\to\mathbb{C}$ be a Lipschitz function. Let $\lambda_1,\lambda_2,\ldots,\lambda_n\in\mathbb{R}$. A divided difference matrix B_f is an $n\times n$ matrix with the following entries:

$$B_f = \left(\frac{f(\lambda_i)-f(\lambda_j)}{\lambda_i-\lambda_j}\right)_{1\leq i,j\leq n}.$$

If the function is differentiable, the entry is equal to $f'(\lambda_i)$ if $i=j$. If the function is not, these entries are set to 0. We will also denote this matrix by $f^{[1]}$, since the entries are a first order divided difference of the λ's.

We want to approximate $\|T_{B_f}\|$. Since this divided difference matrix may appear arbitrary at first, its selection is explained in the following section.

3.4. Divided Difference in Quantum Mechanics

It turns out that this divided difference Schur multiplier has applications in quantum mechanics. To see this, the reader must first be familiar with the commutator of two operators.

Definition 3.16. The commutator of two operators A and B is defined as

$$[A,B] = AB - BA.$$

The commutator measures the extent to which two observables A and B cannot be simultaneously measured with precision. If $[A,B]=0$, then A and B can in fact be measured simultaneously. The commutator plays a crucial role in quantum mechanics as it provides insight into the fundamental structure of quantum observables [5]. In classical mechanics, physical quantities like position and momentum are treated as real numbers that can be measured simultaneously with certainty. However, in quantum mechanics, operators corresponding to these observables do not generally commute. That is, the commutator of these operators is not zero. This means their measurements are subject to an uncertainty.
The properties of a commutator are essential for understanding several key quantum phenomena. For example, the position x and momentum $\hat{p}$ do not commute, which is shown in the canonical commutation relation [5]:

$$[x,\hat{p}] = i\hbar.$$

This relation is key to the Heisenberg uncertainty principle, which states that there exists a fundamental limit to how precisely certain pairs of physical properties of a particle can be known simultaneously. The fact that these quantities do not commute limits how accurately we can know certain pairs of physical properties. Thus to summarize, the commutator shows the core idea of quantum uncertainty.

The following proposition will show how the divided difference matrix comes into play when applying the commutator. Before stating and proving the lemma, the necessary notation is introduced.
Let A,B be self-adjoint matrices in $M_n(\mathbb{C})$ and $f:\mathbb{R}\to\mathbb{C}$ be a function in C^1. Let A be of the form:

$$A = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix},$$

so that $f(A)$ is of the following form:

$$f(A) = \begin{pmatrix} f(\lambda_1) & 0 & \cdots & 0 \\ 0 & f(\lambda_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f(\lambda_n) \end{pmatrix}.$$

Proposition 3.17. Let A, B be as described above. Then:

$$[f(A), B] = T_{B_f}[A, B]$$

Proof.

$$\begin{aligned}
[f(A), B] &= f(A)B - Bf(A) \\
&= \begin{pmatrix} f(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f(\lambda_n) \end{pmatrix} B - B \begin{pmatrix} f(\lambda_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & f(\lambda_n) \end{pmatrix} \\
&= (f(\lambda_i)B_{ij} - B_{ij}f(\lambda_j))_{1\le i,j\le n} \\
&= ((f(\lambda_i) - f(\lambda_j))B_{ij})_{1\le i,j\le n} \\
&= (\frac{f(\lambda_i) - f(\lambda_j)}{\lambda_i - \lambda_j}(\lambda_i - \lambda_j)B_{ij})_{1\le i,j\le n} \\
&= T_{B_f}(((\lambda_i - \lambda_j)B_{ij})_{1\le i,j\le n}) \\
&= T_{B_f}[A, B].
\end{aligned}$$

□

What we see here is that a function can be separated from the commutator by using the Schur multiplier induced by the divided difference matrix. The details of this connection to quantum mechanics are beyond the scope of this thesis, but the aim of this proof is to give the reader an idea as to why the divided difference matrix is chosen for the approximation.

3.5. Objective of Approximation

Combining the Schur multiplier induced by B_f with Equation 3.2, we obtain the following:

$$\|T_{B_f}\| = \sup_{\substack{B\in M_n(\mathbb{C}) \\ \|B\|\neq 0}} \frac{\|T_{B_f}(B)\|_p}{\|B\|_p} = \sup_{\substack{B\in M_n(\mathbb{C}) \\ \|B\|\neq 0}} \frac{\|B_f * B\|}{\|B\|}. \tag{3.3}$$

This is the objective that we want to approximate. The function f that we consider for this approximation is $f(x) = |x|$. The reason for this choice, is it is a simple, yet non-differentiable, Lipschitz function. For this reason, many counterexamples that demonstrate the sharpness or failure of norm bounds are based on the absolute value function. As such, analyzing the Schur multiplier induced by the divided differences of $f(x) = |x|$ provides valuable insight into the limitations and asymptotic behavior of the norm.
Many results have already been obtained concerning this norm, for this function and others, including several bounds that will provide information on the behavior of the norm. First, an important result will be shown for general Schur multipliers.

Proposition 3.18. Let $A = (a_{ij})_{1\le i,j\le n} \in M_n(\mathbb{C})$ and consider $T_A : M_n(\mathbb{C}) \to M_n(\mathbb{C})$. Then:

$$\|T_A\| \ge \max_{i,j} |a_{ij}| \quad \text{for all } 1 \le p \le \infty.$$

Proof. First, recall that $\sup_{B\neq 0} \frac{\|T(B)\|}{\|B\|} = \sup_{\|B\|_p=1} \|T(B)\|$, as we have seen in Remark 2.4. Let (i_0, j_0) be an index such that $|a_{i_0 j_0}| = \max_{i,j} |a_{ij}|$. Define a matrix $B^1 \in M_n(\mathbb{C})$ by

$$B^1_{ij} = \begin{cases} 1 & \text{if } (i,j) = (i_0, j_0), \\ 0 & \text{otherwise.} \end{cases}$$

Then B^1 is a rank-one matrix with a single nonzero entry, and its p-norm satisfies $\|B^1\|_p = 1$ for all $p \in [1,\infty]$. Applying the Schur multiplier to B yields

$$T_A(B^1) = A * B^1,$$

which is a matrix with a single nonzero entry $(A * B^1)_{i_0 j_0} = a_{i_0 j_0}$, and all other entries zero. Thus,

$$\|T_A(B^1)\|_p = |a_{i_0 j_0}| = \max_{i,j} |a_{ij}|.$$

Since $\|B^1\|_p = 1$, it follows that

$$\|T_A\| = \sup_{\|B\|_p = 1} \|T_A(B)\|_p \geq \|T_A(B^1)\|_p = \max_{i,j} |a_{ij}|.$$

□

What we have seen here is that the maximal absolute entry of the Schur multiplier matrix A is a lower bound for the norm on T_A. Since we are considering the absolute value function, we see that the maximal absolute entry of the divided difference matrix B_f is equal to 1. Hence we already know that $\|T_{B_f}\| \geq 1$. Another result that holds for general Schur multipliers is shown in Proposition 3.19.

Proposition 3.19. The space $M_n(\mathbb{C})$ can be naturally embedded as a subspace of $M_{n+1}(\mathbb{C})$, and the Schur multiplier $T_{B_f} : M_n(\mathbb{C}) \to M_n(\mathbb{C})$ can be extended accordingly. Thus, the operator norm of T_{B_f} is monotone non-decreasing as n increases.

Proof. $M_n(\mathbb{C})$ embeds into $M_{n+1}(\mathbb{C})$ via the mapping

$$A \mapsto \tilde{A} := \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix},$$

and the Schur multiplier T_{B_f} can be extended to an operator $T_{B_f^{(n+1)}}$ on $M_{n+1}(\mathbb{C})$ by defining $B_f^{(n+1)} \in M_{n+1}(\mathbb{C})$ such that it agrees with B_f on the top-left $n \times n$ block.
Then, for any $A \in M_n(\mathbb{C})$, we have

$$\|T_{B_f}(A)\|_p = \|T_{B_f^{(n+1)}}(\tilde{A})\|_p \quad \text{and} \quad \|\tilde{A}\|_p = \|A\|_p,$$

which implies

$$\frac{\|T_{B_f}(A)\|_p}{\|A\|_p} \leq \|T_{B_f^{(n+1)}}\|.$$

Taking the supremum over all $A \in M_n(\mathbb{C})$ yields

$$\|T_{B_f}\| \leq \|T_{B_f^{(n+1)}}\|.$$

Hence, the norm of the Schur multiplier is non-decreasing in n. □

In the following section, more key findings are presented that offer insight into the behavior of the norm and provide the reader with an overview of the current state of research in this area.

3.6. Overview of Prior Work

Several important results concerning the operator norm of Schur multipliers have been established in the literature. In particular, the work by Davies [4] provides both upper and lower bounds for the norm of the operator T_{B_f} with f the absolute value function. These results are highly relevant in the context of our study. In his paper, the operator acts on the Schatten classes S_p. In many of the outcomes presented here and throughout the literature, it is assumed that the operator norm on the Schatten classes S_p is finite for any Lipschitz function f. This boundedness is the main result established by Potapov and Sukochev in their 2011 paper [10].

3.6.1. Upper Bounds on the Norm

In Chapter 2 of the paper, Davies [4] shows that for the absolute value function $f(x) = |x|$, the norm of the associated divided difference operator $T_{B_f} : S_p \to S_p$ satisfies the following bound:

$$\|T_{B_f}\| \leq \frac{Cp^2}{p-1}, \quad \text{for } 1 < p < \infty,$$

for some constant $C > 0$. This estimate holds in the context of an infinite-dimensional Hilbert space. The result is a consequence of Proposition 4 and Corollary 5 in the paper [4], which demonstrate that certain kernels of operators with divided differences have norms that can be bounded in terms of the Schatten p-norm of the underlying operator. Due to this result, we see that for $p \to \infty$ the norm will grow as p. Additionally, as $p \downarrow 1$, then the norm will grow as $\frac{1}{p-1}$.

Another important result related to the norm of divided difference Schur multipliers is due to Caspers, Junge, Sukochev, and Zanin. In Theorem 7.6 of their 2019 paper [3], it is shown that Schur multipliers associated with divided differences of Lipschitz functions acting on Schatten classes S_p are bounded with an upper bound that grows at most logarithmically in the matrix dimension. This result generalizes earlier findings that were restricted specifically to $f(x) = |x|$.

In Theorem 5.2 of their paper, Caspers, Montgomery-Smith, Potapov and Sukochev [2] also establish a sharp upper bound for the norm of the Schur multiplier associated with the divided differences of a Lipschitz function. Specifically, they show that for every $1 < p < \infty$, the operator norm $\|T_{B_f}\|_{S_p \to S_p}$ grows at most like $\frac{p^2}{p-1}$. This result confirms that the Schur multiplier defined by B_f for any Lipschitz function is bounded on S_p, and quantifies precisely how this bound depends on p.

Together, these three results reveal a consistent picture of how the norm of the Schur multiplier T_{B_f} behaves across different settings. Davies' result for the specific function $f(x) = |x|$ shows that $\|T_{B_f}\|_{S_p \to S_p} \leq \frac{Cp^2}{p-1}$, $C > 0$, with sharp asymptotic growth as $p \downarrow 1$ and $p \to \infty$. Caspers, Montgomery-Smith, Potapov and Sukochev extend this upper bound to all Lipschitz functions, confirming that the same $\frac{p^2}{p-1}$ growth holds in general, not just for $f(x) = |x|$. In contrast, the result by Caspers, Junge, Sukochev, and Zanin focuses on the dimension dependence and shows that in finite dimensions, the norm of T_{B_f} grows at most logarithmically with matrix size. Together, these findings provide both operator norm and dimensional asymptotics, showing that while the norm can grow significantly with p, it remains controlled with respect to dimension.

3.6.2. Lower Bound on the Norm

In Chapter 3, Davies [4] extends the result mentioned above by giving a lower bound in the finite-dimensional setting for the case $f(x) = |x|$. He constructs explicit examples of self-adjoint matrices A and B in $M_n(\mathbb{C})$ such that the ratio

$$\frac{\|f(A) - f(B)\|_1}{\|A - B\|_1}$$

grows at least like $\log(n)$, where $\|\cdot\|_1$ is the Schatten 1-norm. This shows that the Schur multiplier T_{B_f}, defined by the matrix of divided differences B_f associated with $f(x) = |x|$, cannot have a bound on its norm independent of n. More precisely, Davies proves that

$$\|T_{B_f}\| \geq C \log(n)$$

for some constant $C > 0$.

A selection of the results presented in this section will be used in Chapter 5 to analyze the approximations. In the following chapter, we will introduce and discuss the numerical methods that are implemented to obtain said approximations.

4

Numerical approximation methods

As stated in the previous chapter, we want to approximate the operator norm on a Schur multiplier of a divided difference matrix, i.e. $\|T_{B_f}\|$, where $T_{B_f} : M_n(\mathbb{C}) \to M_n(\mathbb{C})$ is a linear operator from the Banach space $(M_n(\mathbb{C}), \|\cdot\|_p)$ onto itself. The two p-norms that must be computed in order to determine this norm, $\|B_f * B\|_p$ and $\|B\|_p$, can be calculated exactly, given the matrix $B \in M_n(\mathbb{C})$ for which they are optimal. In these calculations, we compute the singular values of the matrices using Remark 3.2.
However, finding this optimal matrix B is not as elementary: the supremum must be approximated. We will do this by approximating the maximum of the norm function $\frac{\|B_f * B\|}{\|B\|}$. There are many ways to do so. A few different approaches will be discussed in the sections that follow. It should be noted that all the code that contains the algorithms described in this chapter are given in Appendix A.

4.1. Brute-Force Method

A rather simple, yet computationally naive way to find the maximum of the norm function is what we will call the Brute-Force method. This algorithm samples random matrices from $M_n(\mathbb{C})$ and evaluates the norm function for each of them, returning the largest value observed as an approximation of the maximum. However, since the input matrices are in $M_n(\mathbb{C})$ they have an infinite amount of possible entries. Consequently, there are infinitely many possible input matrices and computing the norm for every single one is impossible. As a result, evaluating all possible inputs is infeasible, and the likelihood of randomly sampling a matrix that is close to optimal is extremely low. Therefore, this algorithm is computationally expensive and unreliable, making it unsuitable for practical use.

In the following section, we introduce an algorithm that evaluates the norm for a specific class of matrices, rather than relying on multiple randomly generated inputs.

4.2. $A(\ell)$ Method

We will first introduce the matrices of which the norm function is calculated, $A(\ell)$ matrices, before giving the reasoning behind choosing them.

Definition 4.1. Let $\ell \in \mathbb{Z}$. $A(\ell)$ matrices are matrices in $M_n(\mathbb{C})$ with the following entries:

$$A(\ell) = \left(\frac{k_{ij}}{2^\ell}\right)_{1\le i,j\le n}, \quad -2^l \le k \le 2^l, \quad k \in \mathbb{Z}$$

Remark 4.2. To give some insight about the structure of these matrices, consider the case where $\ell = 4$ and $n = 2$. In this setting, the matrices entries are bounded by $-16 \le k \le 16$, which gives 33 possible values for each entry. Since the matrices are 2×2, there are 4 entries in total. Consequently, the total number of possible matrices to consider is $33^4 \approx 1.19 \times 10^6$.

Now, we will provide the reasoning for choosing these matrices. The challenge lies in the fact that the space $M_n(\mathbb{C})$ contains infinitely many matrices, making an exhaustive search for the matrix that maximizes the norm infeasible. The $A(\ell)$ matrices can intuitively be seen as a finite cover of $M_n(\mathbb{C})$. We will see that this

cover is large enough to yield meaningful approximations of the operator norm in general. This result justifies the use of a finite, structured subset of matrices.

To see this, we consider the set $\mathscr{A}_\ell = \{\left(\frac{k_{ij}}{2^\ell}\right)_{1\le i,j\le n} \mid -2^\ell \le k_{ij} \le 2^\ell, k_{ij} \in \mathbb{Z}\}$. Then the following holds:

Theorem 4.3. Let $T_D : M_n(\mathbb{C}) \to M_n(\mathbb{C})$ be a Schur multiplier induced by the matrix $D = (D_{ij})_{1\le i,j\le n} \in M_n(\mathbb{C})$. Suppose that $\forall A \in \mathscr{A}_\ell$ we have:

$$\|T_D(A)\|_p \le C\|A\|_p.$$

Then $\forall B \in M_n(\mathbb{C})$ we have

$$\|T_D(B)\|_p \le C\|B\|_p + \frac{Mn^2 + C}{2^\ell}.$$

Proof. Let $B \in M_n(\mathbb{C})$. Without loss of generality assume that $\|B\| \le 1$. Let

$$A = \left(\frac{[2^\ell B_{ij}]}{2^\ell}\right)_{1\le i,j\le n} \in \mathscr{A}_\ell.$$

Then:

$$\begin{aligned}
\|T_D(B)\|_p &\le \|T_D(A) + T_D(B-A)\|_p \\
&\le \|T_D(A)\|_p + \|T_D(B-A)\|_p \\
&\le C\|A\|_p + \frac{Mn^2}{2^\ell} \\
&= C\|B + A - B\|_p + \frac{Mn^2}{2^\ell} \\
&\le C\|B\|_p + \frac{C}{2^\ell} + \frac{Mn^2}{2^\ell},
\end{aligned}$$

where $M = \max_{i,j} |D_{ij}|$. □

Since the set $\mathscr{A}_\ell$ is finite and the p-norm for all matrices in this set is finite, a C such that the assumption in Theorem 4.3 holds, exists. Hence we know that the Schur multiplier norm is bounded, moreover we obtain an explicit estimate for the bound. Also, we see that choosing a larger value for ℓ will result in a lower fault margin, thus in a more accurate approximation.

Unfortunately, as seen in Remark 4.2, the size of this set increases rapidly with both ℓ and n. As a result, the algorithm becomes computationally expensive and impractical for our purposes. Therefore, we introduce another optimization method. This method, unlike the previous two, does not rely on calculating the norm for various matrices and returning the largest one found, but uses the gradient of the norm function to locate the maximum.

4.3. Gradient Ascent Method

After exploring a brute-force approach, we now turn to a more refined strategy, which we refer to as the gradient ascent method. While the reader may be more familiar with the term gradient descent, used for minimization problems, our goal is to maximize the norm function. Therefore, we employ the gradient descent algorithm in reverse. The idea behind the algorithm is then to ascend along the gradient direction to iteratively approach a maximum. Since the algorithm is designed for minimization problems, we shall discuss the theory behind the algorithm in terms of descent instead of ascent.

Gradient descent is a widely used optimization algorithm for minimizing differentiable functions [8]. It operates by iteratively updating a possible solution in the direction of the negative gradient of the objective function. This corresponds to the direction of steepest descent. At each step, the algorithm moves the current point closer to a minimum by subtracting a scaled version of the gradient. The scaling factor is then known as the learning rate.

Mathematically, given a function $f : \mathbb{R}^n \to \mathbb{R}$ and an initial point $x_0 \in \mathbb{R}^n$, the gradient descent update formula is defined as:

$$x_{k+1} = x_k - \eta \nabla f(x_k),$$

where $\eta > 0$ is the learning rate, and $\nabla f(x_k)$ is the gradient of f at x_k. The process is repeated until convergence, which typically occurs when the gradient becomes sufficiently small or when a fixed number of iterations is reached.

Gradient descent is the foundation for many modern optimization techniques. Its effectiveness depends heavily on the choice of the learning rate and the properties of the objective function, such as convexity and smoothness. Also, when the minimum is approached, the algorithm converges slower, since the gradient gets smaller. In order to speed up the conversion, while also avoiding oscillations, the algorithm can be expanded by implementing two techniques called momentum and root mean square propagation.

4.3.1. Momentum

The momentum method introduces a mechanism inspired by physics. Instead of relying solely on the current gradient, momentum accumulates past gradients to build velocity. This velocity is pointed in the downhill direction. This allows the algorithm to maintain speed in directions of persistent descent and dampen oscillations in directions where the gradient frequently changes sign [13].

Formally, the momentum update formula is given by:

$$v_{k+1} = \beta v_k + (1-\beta)\nabla f(x_k), \quad x_{k+1} = x_k - \eta v_{k+1},$$

where:

- v_k is the velocity (or accumulated gradient),
- $\beta \in [0,1)$ is the decay rate, controlling how much of the previous velocity is retained (typically around 0.9),
- η is the learning rate,
- $\nabla f(x_k)$ is the gradient at step k.

The effect of momentum is to smooth the update trajectory. This allows the optimizer to move more efficiently through curved regions of the loss surface. It often results in faster convergence and improved stability compared to the "simple" gradient descent, also referred to as vanilla gradient descent.

4.3.2. Root Mean Square Propagation

The second technique that decreases convergence time and oscillations is root mean square propagation, also referred to as RMSProp. It is an adaptive learning rate optimization algorithm. In standard gradient descent, a single global learning rate is applied to all parameters, which can lead to inefficiencies, particularly when some gradients are significantly larger than others. RMSProp mitigates this issue by adjusting the learning rate individually for each parameter based on a moving average of its squared gradients. This helps normalize updates across all dimensions of the loss surface, preventing large gradients from forcing a uniformly small learning rate [7].

The RMSProp update formula is given by:

$$s_{k+1} = \rho s_k + (1-\rho)(\nabla f(x_k))^2,$$

$$x_{k+1} = x_k - \frac{\eta}{\sqrt{s_{k+1}} + \varepsilon} \nabla f(x_k),$$

where:

- s_k is the exponentially weighted average of past squared gradients,
- $\rho \in [0,1)$ is the decay rate (typically around 0.9),

- η is the base learning rate,
- $\varepsilon > 0$ is a small constant added for numerical stability,
- $\nabla f(x_k)$ is the gradient at iteration k.

Both of these techniques can also be combined. This merger and more are done in adaptive moment estimation, or Adam optimization.

4.3.3. Adam optimization: combining Momentum and RMSProp

Adam's ability to adapt the learning rate for each parameter, along with its gradient tracking mechanism, results in faster convergence and greater stability during optimization. Simultaneously, it still progresses towards a minimum. These features make Adam particularly well suited for training complex models on large datasets. The update process consists of several key steps:

- **Moment estimates**
  - The first moment estimate accumulates the exponentially decaying average of past gradients:
    $$m_k = \beta_1 m_{k-1} + (1-\beta_1)\nabla f(x_k).$$
  - **T**he second moment estimate keeps track of the exponentially weighted average of the squared gradients:
    $$v_k = \beta_2 v_{k-1} + (1-\beta_2)(\nabla f(x_k))^2.$$
- **Bias Correction**
  Because both m_k and v_k are initialized to zero, their early values are biased toward zero, particularly during the initial iterations. Thus they must be compensated for this initialization bias. To this end, Adam applies the following bias-correction formulas:
  $$\hat{m}_k = \frac{m_k}{1-\beta_1^k}, \quad \hat{v}_k = \frac{v_k}{1-\beta_2^k}.$$
- **Parameter Update Formula**
  Using the corrected estimates, the parameters are updated as follows:
  $$x_{k+1} = x_k - \frac{\alpha}{\sqrt{\hat{v}_k}+\varepsilon}\hat{m}_k.$$

Parameters in Adam optimization

- η is the base learning rate,
- β_1 and β_2 are the decay rates for the first and second moment estimates (default values are $\beta_1 = 0.9$, $\beta_2 = 0.999$),
- ε is a small constant added to the denominator to ensure numerical stability and prevent division by zero.

There are several important differences between simply combining RMSProp with momentum and the Adam optimizer [9].

Firstly, RMSProp with momentum computes parameter updates by applying momentum to the rescaled gradient. Specifically, it first rescales the gradient using a running average of the squared gradients (as in standard RMSProp), and then applies momentum to these adjusted updates. This separation means that the rescaling and the momentum act as two distinct mechanisms.

In contrast, Adam unifies these mechanisms by computing updates directly from a running average of the first moment (the gradient itself) and the second moment (the squared gradient), which are combined in a

single update formula. Importantly, Adam includes bias correction terms for both the first and second moment estimates, compensating for the initialization problem mentioned above.

This bias correction is particularly significant when the second moment decay rate β_2 is set close to 1. Without bias correction, the effective learning rate can become excessively large in early iterations, leading to unstable updates or even divergence. Adam's bias correction mitigates this risk. Also, the objective function we aim to optimize, given by $\frac{\|B_f * B\|}{\|B\|}$, is non-convex due to the presence of both the Schur product and the quotient of norms. This structure can lead to a highly irregular and poorly conditioned optimization landscape, in which small changes in the input matrix B may have inconsistent effects on the norm. In such settings, standard gradient methods often struggle with either vanishing or exploding gradients. However, Adam offers a robust optimization strategy. Among other reasons, this is why Adam optimization is often preferred to the mechanism that merges RMSProp and momentum.

Remark 4.4. As previously noted, our goal is to maximize the norm function. Therefore, throughout the remainder of this text, we will refer to the techniques described in Section 4.3 as gradient ascent methods. We may use these algorithms since maximizing a function can be achieved by minimizing its negation.

In this thesis, Adam gradient ascent is chosen to approximate the supremum of the norm function. The other methods mentioned in Section 4.1 and 4.2 have also been implemented (including a combination of RMSProp and momentum), but they exhibited significantly longer runtimes. Because of this, and Adam's superior efficiency and stability, Adam optimization is chosen as the optimization method. The pseudocode for the implementation is shown in Algorithm 1. The corresponding algorithm for the case $p = \infty$ follows a similar structure and is included, along with the complete implementation details, in Appendix A. In Chapter 5, the results of both these implementations are presented and analyzed.

Algorithm 1 Adam Gradient Ascent for Maximizing Norm Function

function GRADIENTASCENTADAM($B_f, A_{\text{init}}, p, \eta, N, \text{tol}$)
  Initialize $A_{\text{real}} \leftarrow \text{Re}(A_{\text{init}})$ with gradient tracking
  Initialize $A_{\text{imag}} \leftarrow \text{Im}(A_{\text{init}})$ with gradient tracking
  Initialize Adam optimizer on A_{real} and A_{imag} with learning rate η
  $\text{norm}_{\text{last}} \leftarrow 0$
  for step $\leftarrow 1$ to N **do**
    Zero gradients ▷ Reset gradients to 0
    $A \leftarrow A_{\text{real}} + i A_{\text{imag}}$
    $\text{norm} \leftarrow \|B_f * A\|_p / \|A\|_p$
    $f \leftarrow -\text{norm}$ ▷ We minimize the negative to perform maximization
    Backpropagate gradients of f
    if any gradient is NaN or Inf **then**
      raise ValueError
    end if
    Perform optimizer step
    if $|\text{norm} - \text{norm}_{\text{last}}| < \text{tol}$ **then**
      break
    end if
    $\text{norm}_{\text{last}} \leftarrow \text{norm}$
  end for
  $A_{\text{final}} \leftarrow \left(A_{\text{real}} + i A_{\text{imag}}\right)$
  return norm, $A_{\text{final}} / \|A_{\text{final}}\|_p$
end function

5

Results and Analysis

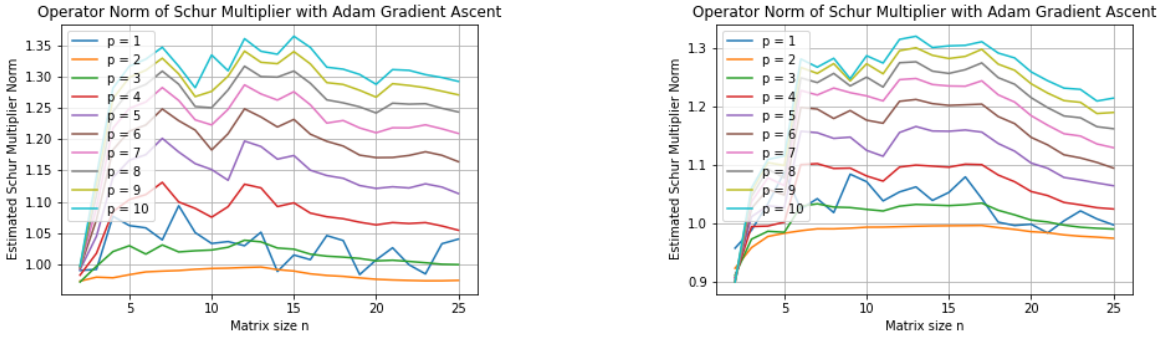
Based on the findings in Chapter 4, we approximate the operator norm by implementing the Adam method. Using Algorithm 1 as presented in the previous chapter, we compute and visualize the estimated norm function both as a function of the matrix size n and of the parameter p. The specific code that is carried out can be found in Appendix A. Based on these results we shall analyze the behavior of the norm function as these parameters increase. In order to implement the code, we employ Python 3. The Torch package is used, since it contains an Adam command that performs the update formulas presented in Chapter 4. Additionally, the Time package is employed to compare the runtimes of different implementations and the MatPlotLib package enables us to visualize the results in several plots.

Remark 5.1. In all approximations, $f(x) = |x|$ is used to create the divided difference matrix, and the lambdas are chosen randomly in $[-10, 10] \subset \mathbb{R}$.

We will treat the case $p = \infty$ separately, as this p-norm, and thus the operator norm as well, is defined differently from the p-norm for $p < \infty$.

5.1. Norm as a Function of n

First, we will consider the norm function as a function of n. By iterating over several values of p, the behavior for different p can be seen simultaneously. To start, the initial matrix that is used in the algorithm is a complex matrix generated randomly with the entries $(a_{ij} + b_{ij} \cdot i)_{1 \leq i,j \leq n}$, where $-10 \leq a_{ij}, b_{ij} \leq 10$. In order to have a manageable runtime, the norm is calculated for $n \in \{2, \ldots, 25\}$, for $p \in \{1, \ldots, 10\}$. The code that is run to generate these plots is indicated by A.1 in Appendix A in the comments (green).



(a) Runtime: 30.6 min.

(b) Runtime:43.8 min.

Figure 5.1: Norm as a function of matrix size n with random initial matrix.

Since the initial matrix is chosen at random, just as the λ's, the results may vary with every run. With this in mind, two runs of the same algorithm are shown in Figure 5.1. As established in Proposition 3.18, the norm is

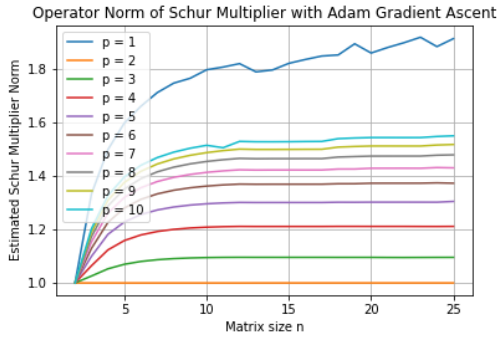
theoretically bounded below by the maximal absolute entry value of the Schur multiplier, which in this case is equal to 1. In the graphs presented in Figure 5.1, we observe that the approximated values of the norm generally reflect this lower bound. Occasional deviations below 1 can be attributed to numerical error due to the approximation method. Thus, this result is overall in line with expectations.
In addition to this, we know by Proposition 3.19 that the norm should not decrease as n increases. However, both in Figure 5.1a and Figure 5.1b, a decrease in the norm is visible. This deviation suggests that the observed values may not reflect the true supremum of the norm function. One possible explanation is a flaw in the implementation or numerical instability. Another likely cause is the random initialization of the input matrix in the gradient ascent algorithm. Since the norm function being optimized is not known to be convex, the optimization may converge to a local maximum rather than the global one. Consequently, the norm estimates could fall below their true values, especially for larger n where the optimization landscape may be more complex. Therefore, we consider another initial matrix.

5.1.1. Introducing a Different Initial Matrix

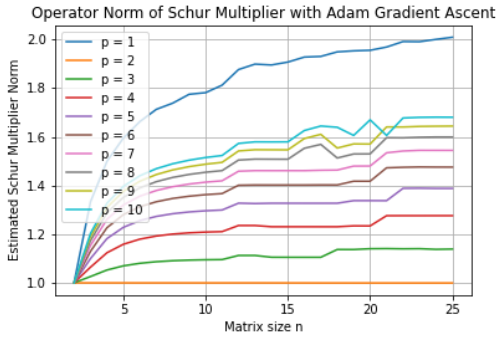
A natural choice for the initial matrix in the gradient ascent algorithm is the matrix with all entries equal to 1. In particular, Davies [4] uses this matrix in the construction of sharp lower bounds for finite p, strongly suggesting that it lies close to a maximizer of the norm function. Thus it provides a good starting point for our gradient ascent algorithm. The results of the approximated norm with the all-1 matrix as our starting point are shown in Figure 5.2.

Remark 5.2. It should be noted that the effectiveness of the all-1 matrix in numerical optimization depends on the specific problem setup and the objective function. In particular, while this matrix may be close to a maximizer of the norm function in certain contexts, it is not always guaranteed to provide the best starting point for every problem.

Remark 5.3. In order to ensure numerical stability, a small perturbation (with order of magnitude 10^{-4}) is added to the entries of the initial matrix.



(a) Runtime: 17.9 min.



(b) Runtime:19.2 min.

Figure 5.2: Norm as a function of matrix size n with all-1 initial matrix.

Once again, the choice of the λ's introduces a randomness to the result, which is why two plots are shown to illustrate the variety of results. Immediately, we see that the resulting plots of the approximated norm are smoother. This can be explained by the fact that the all-1 matrix provides a more structured and symmetric starting point. In contrast to random initialization, which may lead to convergence to different local maxima, the all-1 matrix ensures more consistent optimization results. In addition, we see that the graphs now in fact do not decrease as n increases. Thus, we indeed see that the optimization process becomes more stable and reliable with this initial matrix.

However, another factor that introduces randomness into the numerical results, as mentioned above, is the choice of the λ's used to construct the divided difference matrix associated with the function $f(x) = |x|$. Since these values vary between runs, the resulting Schur multiplier matrix varies as well, which complicates the interpretation of the results. To reduce this randomness and obtain a more stable and interpretable approxi-

mation, we look for an alternative that preserves the key structural features of the divided difference matrix. We therefore introduce the sign matrix B_{sgn}.

5.1.2. Introducing the Sign Matrix as Schur Multiplier

The sign matrix B_{sgn} is defined by:

$$(B_{sgn})_{ij} = \begin{cases} 1 & \text{if } i \leq j, \\ -1 & \text{if } i > j. \end{cases}$$

This matrix naturally arises from the limiting behavior of the divided difference matrix B_f when choosing specific values for the λ's, namely $\lambda_i = \frac{1}{2^{ki}}$ for a large integer k. In this case, we see that as $k \to \infty$, the entries of the divided difference matrix tend to the entries of the sign matrix. This connection provides a deterministic approximation of the divided difference matrix for $f(x) = |x|$. Therefore, we present numerical results obtained by using this sign matrix as the Schur multiplier in the gradient ascent algorithm in Figure 5.3.

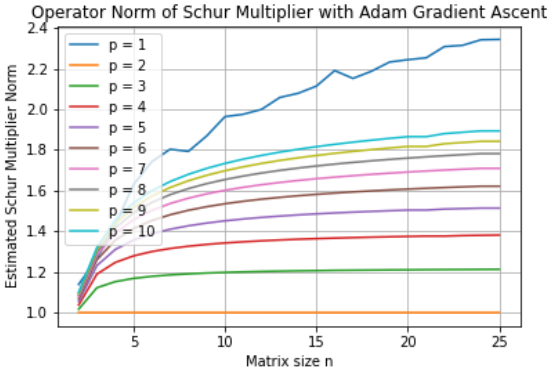


Figure 5.3: Norm as a function of matrix size n with all-1 initial matrix and B_{sgn} as Schur multiplier. Runtime: 26.1 min.

The resulting plot exhibits an even smoother and more stable behavior compared to the previous results obtained with the divided difference matrix. This improvement can likely be attributed to the structure of the sign matrix, which avoids variability. Importantly, the initial matrix A_{init}, consisting of all entries equal to one with a small perturbation, is still being used. The similarity in the overall shape of the norm function when using the the sign matrix and when using the divided difference matrix further supports the idea that B_{sgn} serves as a meaningful and stable approximation of B_f, particularly in the context of numerical optimization. These observations justify the use of the sign matrix to explore the behavior of the norm function further.

An additional observation is that the computed norm values in the final implementation are slightly larger than those obtained before. This can be attributed to the fact that the sign matrix contains only values in $\{-1, 1\}$, which are the extreme values the divided difference matrix can attain. This means the sign matrix may slightly overestimate the magnitude of certain entries in the divided difference matrix, especially when the λ's are not close to zero. This effect offers a reasonable explanation for the observed increase.

In addition to the norm as a function of the matrix size n, we are also interested in the behavior of the norm as a function of the parameter p. This will be discussed in the next section.

5.2. Norm as a function of p

To examine the behavior of the norm as a function of p, a slightly different plotting approach is used. Since our interest lies in understanding the behavior for large matrix dimensions, we first fix a relatively large value of n. We then plot the calculated norm as a function of p, keeping n constant. This code can be found under A.2 in Appendix A and the result is presented in Figure 5.4.

Based on the theoretical results obtained by Davies [4] that are presented in Section 3.6, we can form expectations about the behavior of the norm function. In particular, as seen before, the norm is at most proportional to $\frac{p^2}{p-1}$ for all $p > 1$. This implies that the norm increases rapidly as $p \downarrow 1$, and grows linearly in p as $p \to \infty$.

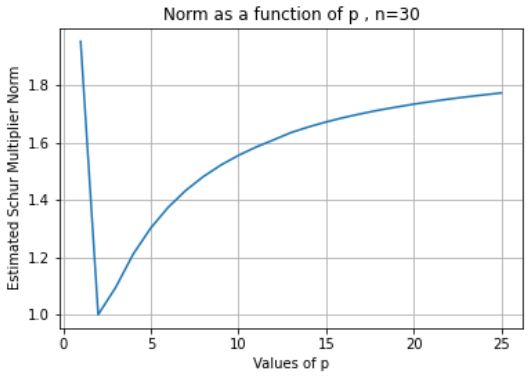


Figure 5.4: Norm as a function of p for $n = 30$ with all-1 initial matrix and B_f as Schur multiplier. Runtime: 5.4 min.

Thus, these results suggest that the norm function should be non-decreasing in p, with a pronounced growth near $p = 1$ and a more gradual increase for large p. However, Davies proved this for infinite dimensions, thus since we take n fixed, we do not expect to see linear growth.

When observing the resulting graph, we indeed see a steep increase in the norm as $p \downarrow 1$, consistent with the theoretical expectation. A gradual growth is also visible for large p. For comparison, we again implement B_{sgn} as the Schur multiplier. This result is shown in Figure 5.5.

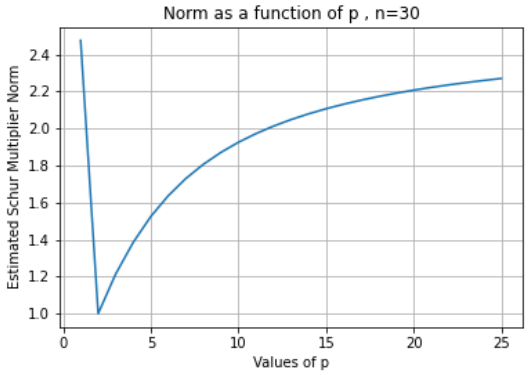


Figure 5.5: Norm as a function of p for $n = 30$ with all-1 initial matrix and B_{sgn} as Schur multiplier. Runtime: 6.1 min.

Once again, we observe an overall increase in the norm values, while the qualitative behavior of the function remains largely unchanged. Given the close similarity between the plots obtained using the sign matrix and those using the divided difference matrix, the substitution does not appear to give significant additional insight in this context. We will continue with the sign matrix as Schur multiplier, since it is computationally less expensive.

In order to obtain a more accurate approximation of the norm function, we modify the approach: instead of computing the norm for a fixed matrix size n, we evaluate it over several values of n and take the supremum. By optimizing over multiple dimensions, we reduce the risk of underestimating the norm due to a suboptimal matrix size, and so achieve a closer approximation to the true supremum defining the operator norm. The result of this alteration is presented in Figure 5.6, of which the code can be found under A.3 in Appendix A.

When taking the supremum over all n, we expect to see linear growth. However, since this is not possible in a reasonable runtime, the supremum is taken over $n \leq 30$. This explains why in Figure 5.6, the linear increase in p is again not visible. There are also a few other factors that probably influence the result. First, the norm might still converge to a local maximum. Additionally, numerical effects, such as precision errors or instabil-

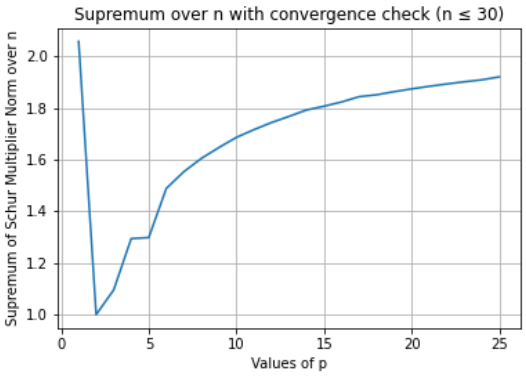


Figure 5.6: Norm as a function of p with all-1 initial matrix and B_{sgn} as Schur multiplier, obtained by taking the supremum over $n \leq 30$. Runtime: 73.8 min.

ity in the optimization process, could cause errors in the result.

What is also interesting when computing the norm by finding this supremum, is observing for which matrix size n this supremum is found. Or, in other words, when the algorithm converges. As can be seen in the code in Appendix A, this convergence happens when the difference between two norms for $n = k$ and $n = k+1$ is less than a predetermined tolerance, three times in a row. This means that the norm increases very little or does not increase anymore when n increases. Therefore, we take the norm as converged. The values of n that are found, are presented in Table B.1 in Appendix B.

From the values in this table we derive that the norm function often converges at $n = 25$ for larger values of p. This is to some extent in line with the results shown in Figures 5.1 - 5.3, since we observe a diminishing increase in the estimated norm. These results are consistent with the theoretical expectation that the operator norm of Schur multipliers becomes asymptotically stable as both n and p increase.

5.3. Norm for $p = \infty$

As mentioned above, we will now observe the operator norm for $p = \infty$. Similarly as in Figures 5.1 - 5.3, the estimated operator norm is plotted as a function of the matrix size n. Before, we used the all-1 initial matrix. However, the reason for this was that the matrix was used in construction of lower bounds for finite p [4]. Now that we take $p = \infty$, this initial matrix is no longer a logical choice, and we return to the usage of random initial matrices with entries $(a_{ij} + b_{ij} \cdot i)_{1 \leq i,j \leq n}$, where $-10 \leq a_{ij}, b_{ij} \leq 10$. This code is given under A.4 in Appendix A. The result is shown in Figure 5.7.

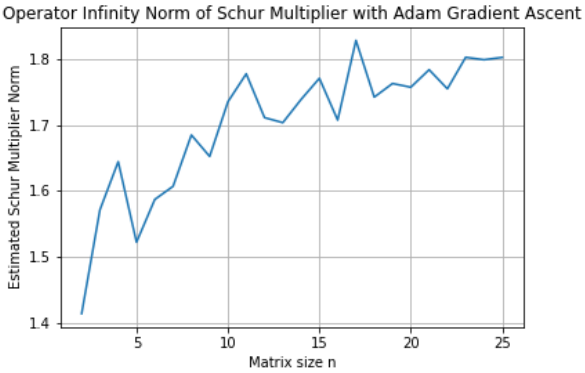


Figure 5.7: Norm for $p = \infty$ as a function of n with random initial matrix and B_{sgn} as Schur multiplier. Runtime: 2.6 min.

Once again returning to Davies' paper [4], we see that he also provides a dimension-dependent lower bound

in Chapter 3, showing that for any fixed p, the norm must grow at least as fast as $\log(n)$ with the matrix dimension n. In the graph of the estimated norm shown in Figure 5.7, we already roughly see the logarithmic growth of the norm function. To determine how this growth will behave for larger p, we fit a logarithmic function of the form $g(n) = s + t\log(n)$, $s, t \in \mathbb{R}$, to the obtained data, using least squares estimation. This method is introduced in the following section.

5.3.1. Least Squares Estimation

The method of least squares estimation is a statistical technique used to determine the best-fitting curve to a set of data points by minimizing the sum of the squared differences between the observed values and the values predicted by the model. In this case, we use the fitting function $g(n) = s + t\log(n)$ where s and t are the parameters to be estimated. This model assumes that the relationship between the independent variable n and the dependent variable is approximately linear when the logarithm of n is taken, which is based on the proven lower bound mentioned above. To perform the least squares estimation, we calculate the difference between the observed norm values and the predicted values $g(n)$, square these differences, and minimize the sum of these squared residuals. The resulting values of s and t provide the best logarithmic fit to the data, capturing the asymptotic behavior of the norm as a function of n. The result of the least squares estimation is shown in Figure 5.8, and the fitted function is $g(n) = 1.3796 + 0.1336\log(n)$.

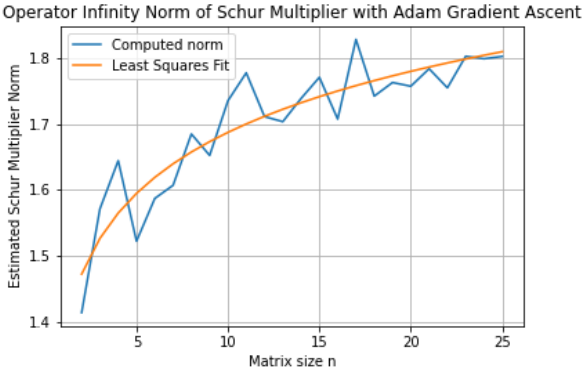


Figure 5.8: Least squares estimation for infinity norm. Runtime: 2.6 min.

We see that the plot matches the logarithmic curve rather well. Still, there are deviations, that we can again attribute to the randomness of the initial matrix and numerical error. In order to analyze this plot further, we observe the residuals for each value of n. An overview of the residuals can be found in Table B.2 in Appendix B. The residuals are all located in $[-0.08, 0.08]$, thus stay rather small. This indicates that the fitted curve closely matches the data. Moreover, the residuals appear to be randomly distributed, with no discernible pattern, which is characteristic of a well-fitting model. This randomness implies that the errors are unbiased and evenly spread across the entire range of n, further confirming the quality of the fit. These observations combined suggest that the fitted function $g(n) = 1.3796 + 0.1336\log(n)$ provides a good approximation to the estimated norm.

In this chapter, we have examined the operator norm of Schur multipliers, both as a function of the matrix size n and the parameter p. Through the implementation of gradient ascent and the use of different initial matrices, we observed how the norm stabilizes and converges with increasing matrix size, largely in line with theoretical expectations. By fitting logarithmic functions to the norm data, we captured the asymptotic growth of the norm, the results showing consistency with the lower bounds derived from Davies' work [4]. Additionally, the choice of the sign matrix as the Schur multiplier provided a stable approximation, which allowed for smoother results. These observations offer a solid foundation for the next chapter, where we will conclude the study and present the key findings.

6

Conclusion

This thesis has investigated the operator norm of Schur multipliers induced by divided differences, with particular focus on the function $f(x) = |x|$ in the space $M_n(\mathbb{C})$. The primary goal was to approximate the operator norm $\|T_{B_f}\|$ numerically, as defined by Equation 1.1, and to analyze its behavior as a function of the Schatten norm parameter p and the matrix size n. By approximating this supremum numerically, the study aimed to provide insights into the behavior of the norm in finite dimensions, as well as the effectiveness of various implementations of an optimization method.

To achieve this, several numerical methods were explored, including brute-force sampling, the use of structured matrices $A(\ell)$, and gradient ascent algorithms. Among these, the Adam optimization algorithm looked to be the most effective in dealing with the non-convex nature of the norm function. Adam combines momentum and RMSProp, two optimization techniques that are well suited for the task of maximizing the norm function. Using Adam in Python, the operator norm was successfully approximated for various values of p and matrix sizes n.

The numerical experiments confirmed several theoretical predictions and provided a deeper understanding of how the norm behaves in practice. Specifically, the results showed that the operator norm tends to increase as $p \downarrow 1$, as expected from the theoretical analysis. Furthermore, the norm exhibits stability for larger matrix sizes n. These results are consistent with previous research on the upper and lower bounds for Schur multipliers and provide further evidence that the selected optimization method, Adam optimization, can effectively handle the complexities of this problem.

One important observation from the numerical analysis is the convergence of the norm as n increases, with certain matrix sizes yielding stable values for the operator norm. This convergence is crucial as it supports the idea that the norm stabilizes for larger matrix sizes. However, there were some deviations from theoretical expectations in certain cases. These appear likely due to the non-convexity of the optimization landscape, which led to local maxima instead of the global maximum in some runs. The use of different initial matrices, particularly random and all-1 matrices, also influenced the results, with the all-1 matrix providing more consistent optimization results.

The findings in this thesis also highlight the importance of structured matrices, like the sign matrix B_{sgn} in obtaining stable and reliable results when approximating the norm. This matrix, derived from the limiting behavior of divided difference matrices, offers a deterministic and computationally less expensive alternative to random initialization. Its use in the gradient ascent algorithm further confirmed the advantages of structured approaches in numerical optimization.

Looking forward, several possibilities for future research arise from the work presented here. One possible direction is to explore the behavior of Schur multipliers in infinite-dimensional settings, where the norm calculations and the background theory become more complex. Additionally, investigating the development of more sophisticated optimization techniques could further reduce the risk of converging to local maxima and improve the accuracy of the approximations. Perhaps the most intriguing step forward, however, lies in the

study of multilinear Schur multipliers. Many results that are proven for the linear case, a few of which were presented in this work, are questions yet to be answered in the multilinear case. This is argued in more detail in the Discussion.

In conclusion, this thesis has validly approximated the operator norm of Schur multipliers in finite-dimensional spaces, providing valuable insights into its behavior and the effectiveness of the Adam optimization algorithm in this context. The results contribute to the broader understanding of Schur multipliers and their role in operator theory and quantum mechanics, and open up opportunities for further exploration in both finite and infinite-dimensional settings.

7

Discussion

In this section, we discuss several aspects related to the methodology, assumptions, and limitations encountered during the work on this thesis. We also explore areas for further investigation, particularly those that extend beyond the scope of this study.

7.1. Assumptions Made in the Code

The selection of the values $\lambda_1, \ldots, \lambda_n$ in the range $[-10, 10]$ is somewhat arbitrary but was chosen to avoid extreme values that might introduce numerical instability or outliers. The values are distributed randomly within this range to prevent any bias that could influence the results. While this range is chosen to be neither too close to 0 nor too far, a more optimized selection could be explored in future studies to better understand its impact on the results.

Similarly, the random choice of matrix entries in the range $[-10, 10]$ for the initial matrix was chosen without a specific guiding principle. The reasoning behind this choice is that the norm is ultimately normalized, so the exact starting values of the entries in the matrix do not significantly influence the final outcome. That said, different initializations could be explored, even more than already done in this work, to assess their impact on the optimization process.

The choice of a learning rate of 0.001 for the Adam optimization algorithm is fairly standard in many optimization problems. While this value is commonly used in practice, it is not necessarily optimal for all cases. Fine-tuning the learning rate could improve the convergence speed and accuracy of the optimization. However, since in Adam optimization the learning rate is adapted per run, the impact of the initial learning rate is already less than in an algorithm with a constant learning rate. Still, future research could explore the effect of different initial learning rates.

7.2. Limitations of the Adam Optimization Algorithm

While the Adam optimization algorithm has proven to be effective for approximating the operator norm in this study, it does come with some limitations. The primary drawback of Adam, and gradient-based optimization methods in general, is its tendency to converge to local minima in non-convex optimization landscapes. Despite the combination of momentum and RMSProp helping to lessen this issue, the optimization can still be sensitive to initialization.
Additionally, the algorithm's performance heavily depends on the choice of hyperparameters such as the learning rate and the decay rates β_1 and β_2. In some cases, improper parameter tuning can lead to slower convergence or suboptimal solutions. While Adam is relatively robust, there is still room for improvements or alternative algorithms. Exploring methods that adapt to the problem's specific characteristics could be worthwhile.

7.3. The Multilinear Case

Finally, as mentioned in the conclusion, one of the most exciting directions for future research lies in the consideration of multilinear Schur multipliers. While this thesis has focused on the linear case, the multilinear case is where the truly challenging questions lie. Many results that have been proven for linear Schur multipliers are yet to be established for multilinear versions of these operators. This opens up a vast area of research that could have implications for operator theory, functional analysis, and applications to quantum mechanics.

In particular, the results shown in Figure 5.6 and 5.7 are open questions in multilinear settings. Since Adam optimization has proven to be an effective method on this subject, a possible next step could be to adapt and apply this method to multilinear operators. This would involve both theoretical and computational exploration, making it a promising direction for future work.

Bibliography

[1] N. L. Carothers. *Real Analysis*. Cambridge University Press, Cambridge, 1st edition, 2000. ISBN 978-0521782573.

[2] M. Caspers, S. Montgomery-Smith, D. Potapov, and F. Sukochev. The best constants for operator lipschitz functions on schatten classes. *Journal of Functional Analysis*, 267(10):3557–3579, 2014. ISSN 0022-1236. doi: https://doi.org/10.1016/j.jfa.2014.08.018. URL https://www.sciencedirect.com/science/article/pii/S0022123614003450.

[3] M. Caspers, M. Junge, F. Sukochev, and D. Zanin. Bmo-estimates for non-commutative vector valued lipschitz functions. *Journal of Functional Analysis*, 278(3):108317, 2020. ISSN 0022-1236. doi: https://doi.org/10.1016/j.jfa.2019.108317. URL https://www.sciencedirect.com/science/article/pii/S0022123619303118.

[4] E. B. Davies. Lipschitz continuity of functions of operators in the Schatten classes. *J. London Math. Soc. (2)*, 37(1):148–157, 1988. ISSN 0024-6107,1469-7750. doi: 10.1112/jlms/s2-37.121.148. URL https://doi.org/10.1112/jlms/s2-37.121.148.

[5] D. J. Griffiths and D. F. Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Press, Cambridge, 3rd edition, 2018. ISBN 9781107189638. URL https://www.cambridge.org.

[6] T. Hytönen, J. van Neerven, M. Veraar, and L. Weis. *Analysis in Banach Spaces, Volume I: Martingales and Littlewood-Paley Theory*, volume 63 of *Ergebnisse der Mathematik und ihrer Grenzgebiete. 3. Folge / A Series of Modern Surveys in Mathematics*. Springer, Cham, 2016. ISBN 978-3-319-31245-1. doi: 10.1007/978-3-319-31246-8.

[7] P. Kashyap. Understanding rmsprop: A simple guide to one of deep learning's powerful optimizers, 2020. URL https://medium.com/@piyushkashyap045/understanding-rmsprop-a-simple-guide-to-one-of-deep-learnings-powerful-optimizers-403baeed9922.

[8] Khan Academy. What is gradient descent? https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent, n.d.

[9] D. P. Kingma and J.L. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/pdf/1412.6980.

[10] D. Potapov and F. Sukochev. Operator-Lipschitz functions in Schatten-von Neumann classes. *Acta Math.*, 207(2):375–389, 2011. ISSN 0001-5962,1871-2509. doi: 10.1007/s11511-012-0072-8. URL https://doi.org/10.1007/s11511-012-0072-8.

[11] I.G. Todorov and L. Turowska. Schur multipliers in operator algebras and harmonic analysis. *arXiv*, 2009. URL https://arxiv.org/pdf/0911.0606.pdf.

[12] J. van Neerven. *Functional Analysis*. European Mathematical Society, Zuerich, 2010. ISBN 9783037190156. URL https://www.ems-ph.org/books/book.php?proj_nr=128.

[13] R. Varma. Gradient descent with momentum, 2019. URL https://towardsdatascience.com/gradient-descent-with-momentum-59420f626c8f/.

[14] G. Wildschut. Approximation of the operator norm of schur multipliers using gradient ascent methods. July 2025.

A

Python code

```python
import torch
import numpy as np
import matplotlib.pyplot as plt
import time

### Calculate the p-norm of a matrix A
def p_norm(A, p):
    if not torch.isfinite(A).all():
        raise ValueError("Non-finite values detected in input matrix A")
    A_star = A.conj().T
    A_star_A = A_star @ A
    _, S, _ = torch.linalg.svd(A_star_A.float())
    S_sqrt = torch.sqrt(S)
    return torch.sum(S_sqrt**p).real**(1/p)

### Calculate the infinity norm of a matrix A
def p_norm_infty(A):
    A_star = A.conj().T
    A_star_A = A_star @ A
    _, S, _ = torch.linalg.svd(A_star_A)
    S_sqrt = torch.sqrt(S)
    S_sorted, _= torch.sort(S_sqrt)
    return S_sorted[-1]

### Construction of B_sgn
def b_sgn(n):
    B = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if i <=j:
                B[i,j]=1
            else:
                B[i,j]=-1
    return torch.tensor(B, dtype=torch.complex128)

### Construction of B_f
def bf(f, lambdas):
    n = len(lambdas)
    B = np.zeros((n, n), dtype=np.complex128)
    for i in range(n):
        for j in range(n):
            if i != j:
                B[i, j] = (f(lambdas[i]) - f(lambdas[j])) / (lambdas[i] - lambdas[j])
    return torch.tensor(B, dtype=torch.complex128)

### Construction of lambdas
def lambdas_set(num_elements, lower_bound=-10, upper_bound=10):
    lambdas_tot = set()
    while len(lambdas_tot) < num_elements:
        lambdas_tot.add(np.random.uniform(lower_bound, upper_bound))
```

```
    return list(lambdas_tot)

### Gradient Ascent algorithm with Adam Optimization
def gradient_ascent_adam(B_f, A_init, p, learning_rate=0.001, num_steps=5000, tol=1e-6):
    A_real = A_init.real.clone().detach().requires_grad_(True)
    A_imag = A_init.imag.clone().detach().requires_grad_(True)
    optimizer = torch.optim.Adam([A_real,A_imag], lr=learning_rate)
    last_val = 0.0
    for step in range(num_steps):
        optimizer.zero_grad()
        A = torch.complex(A_real,A_imag)
        norm_A = p_norm(A, p)
        A_normalized = A / norm_A
        BfA = B_f * A_normalized
        norm_val = p_norm(BfA, p)
        loss = -norm_val
        loss.backward()
        torch.nn.utils.clip_grad_norm_([A_real, A_imag], max_norm=20)
        if not torch.isfinite(A_real.grad).all() or not torch.isfinite(A_imag.grad).all():
            raise ValueError(f"NaN or Inf detected in gradients at step {step}")
        optimizer.step()
        if abs(norm_val.item() - last_val) < tol:
            break
        last_val = norm_val.item()
    A_final = torch.complex(A_real, A_imag)
    A_final = (A_final / p_norm(A_final, p)).detach()
    return norm_val.item(), A_final

### Gradient Ascent algorithm with Adam Optimization for p = \infty
def gradient_ascent_adam_infty(B_f, A_init, learning_rate=0.001, num_steps=5000, tol=1e-6):
    A = A_init.clone().detach().requires_grad_(True)
    optimizer = torch.optim.Adam([A], lr=learning_rate)
    last_val = 0.0
    for step in range(num_steps):
        optimizer.zero_grad()
        norm_A = p_norm_infty(A)
        A_normalized = A / norm_A
        BfA = B_f * A_normalized
        norm_val = p_norm_infty(BfA)
        loss = -norm_val
        loss.backward()
        optimizer.step()
        if not torch.isfinite(A.grad).all():
            raise ValueError(f"NaN or Inf detected in gradients at step {step}")
        if abs(norm_val.item() - last_val) < tol:
            break
        last_val = norm_val.item()
    A_final = (A / p_norm_infty(A)).detach()
    return norm_val.item(), A_final

### A.1: Plotting Norm vs matrix size n, multiple fixed p.
### The code is currently shown for computing the norm with a random initial matrix and B_f as Schur multiplier.
### Uncomment lines in the loop to obtain code for all-1 initial matrix and B_sgn as Schur multiplier.
if __name__ == "__main__":
    start = time.time()
    pmax = 10
    pvalues = np.arange(1, pmax+1)
    nmax = 25
    f_abs = lambda x: abs(x)
    lambdas = lambdas_set(nmax)
    nlist = np.arange(2, nmax + 1)
    real_part = (torch.rand(nmax, nmax) * 20) - 10
    imag_part = (torch.rand(nmax, nmax) * 20) - 10
    A_base = (real_part + 1j * imag_part).to(torch.complex128)
    for p in pvalues:
        res = np.zeros(nmax - 1)
```

```
        for idx, n in enumerate(nlist):
            B_f = bf(f_abs, lambdas[:n])
            eps = 1e-4
            # A_init = torch.ones((n,n), dtype = torch.complex128)+eps * (torch.randn(n, n) + 1j * torch.randn(n, n))
            A_init = A_base[:n,:n]
            res[idx], _ = gradient_ascent_adam(B_f, A_init, p)
            # res[idx], _ = gradient_ascent_adam(b_sgn(n), A_init, p)
            print(f'p={p}, n={n}, norm={res[idx]:.4f}')
        plt.plot(nlist, res, label=f'p = {p}')
    plt.xlabel('Matrix size n')
    plt.ylabel('Estimated Schur Multiplier Norm')
    plt.title('Operator Norm of Schur Multiplier with Adam Gradient Ascent')
    plt.legend()
    plt.grid(True)
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    plt.savefig(f'plot1_n{nmax}_p{pmax}_gaa_fin_{timestamp}.png')
    plt.show()
    print("Runtime: "+str((time.time()-start)/60)+" min")

### A.2: Plotting Norm vs p, one fixed n
### The code is currently shown for computing the norm with the all-1 initial matrix and B_f as Schur multiplier.
### Uncomment lines in the loop to obtain code for B_sgn as Schur multiplier.
if __name__ == "__main__":
    start = time.time()
    f_abs = lambda x: abs(x)
    n = 30
    pmax = 25
    plist = np.arange(1,pmax+1)
    lambdas = lambdas_set(n)
    B_f = bf(f_abs,lambdas)
    real_part = (torch.rand(n, n) * 20) - 10
    imag_part = (torch.rand(n, n) * 20) - 10
    # A_init = (real_part + 1j * imag_part).to(torch.complex128)
    eps = 1e-4
    A_init = torch.ones((n,n),dtype=torch.complex128) +eps * (torch.randn(n, n) + 1j * torch.randn(n, n))
    res = np.zeros(pmax)
    for p in range(1, pmax+1):
        try:
            res[p-1], _ = gradient_ascent_adam(B_f, A_init, p)
            # res[p-1], _ = gradient_ascent_adam(b_sgn(n), A_init,p)
            print(f'p={p}, norm = {res[p-1]:.4f}')
        except Exception as e:
            print(f'Failed at p={p}: {e}')
            res[p-1] = np.nan
    plt.plot(plist,res)
    plt.xlabel('Values of p')
    plt.ylabel('Estimated Schur Multiplier Norm')
    plt.title(f'Norm as a function of p , n={n}')
    plt.grid(True)
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    plt.savefig(f'plot2_n{n}_p{pmax}_gaa_{timestamp}.png')
    plt.show()
    print("Runtime: {:.2f} min".format((time.time() - start) / 60))

### A.3: Plotting norm vs p, taking the supremum over n
if __name__ == "__main__":
    start = time.time()
    f_abs = lambda x: abs(x)
    pmax = 25
    nmax = 30
    tol = 1e-5
    patience = 3
    plist = np.arange(1, pmax + 1)
    res = np.zeros(pmax)
    lambdas = lambdas_set(nmax)
    for p in range(1, pmax + 1):
        norm_max = -np.inf
        no_improve_count = 0
```

```
        for n in range(2, nmax + 1):
            try:
                B_f = bf(f_abs, lambdas[:n])
                eps = 1e-4
                A_init = torch.ones((n,n), dtype = torch.complex128) + eps * (torch.
    randn(n, n) + 1j * torch.randn(n, n))
                norm_val, _ = gradient_ascent_adam(b_sgn(n),A_init,p)
                if norm_val > norm_max + tol:
                    norm_max = norm_val
                    no_improve_count = 0
                else:
                    no_improve_count += 1

                if no_improve_count >= patience:
                    print(f'p={p}: converged at n={n}')
                    break
            except Exception as e:
                print(f"Failed at p={p}, n={n}: {e}")
                continue
        res[p-1] = norm_max
        print(f'p={p},sup_n ||T_Bf|| ~ {norm_max:.4f}')
    plt.plot(plist, res)
    plt.xlabel('Values of p')
    plt.ylabel('Supremum of Schur Multiplier Norm over n')
    plt.title(f'Supremum over n with convergence check (n \leq {nmax})')
    plt.grid(True)
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    plt.savefig(f'plot_sup_n_to_{nmax}_p{pmax}_conv_{timestamp}.png')
    plt.show()
    print("Runtime: {:.2f} min".format((time.time() - start) / 60))

### A.4: Plotting infinity norm vs. p
if __name__ == "__main__":
    start = time.time()
    nmax =25
    f_abs = lambda x: abs(x)
    lambdas = lambdas_set(nmax)
    nlist = np.arange(2, nmax + 1)
    res = np.zeros(nmax-1)
    for idx, n in enumerate(nlist):
        B_f = bf(f_abs,lambdas[:n])
        real_part = (torch.rand(n, n) * 20) - 10
        imag_part = (torch.rand(n, n) * 20) - 10
        A_init = (real_part + 1j * imag_part).to(torch.complex128)
        res[idx], _ = gradient_ascent_adam_infty(b_sgn(n), A_init)
        print(f'n={n}, norm = {res[idx]}')
    plt.plot(nlist,res, label='Computed norm')
    plt.xlabel('Matrix size n')
    plt.ylabel('Estimated Schur Multiplier Norm')
    plt.title('Operator Infinity Norm of Schur Multiplier with Adam Gradient Ascent')
    plt.grid(True)
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    plt.savefig(f'plot4_n{nmax}_gaa_{timestamp}.png')
    plt.show()
    plt.plot(nlist,res, label='Computed norm')
    log_n = np.log(nlist)
    A_ls = np.vstack([np.ones_like(log_n),log_n]).T
    x_ls, residuals, rank, s = np.linalg.lstsq(A_ls,res,rcond=None)
    S, T = x_ls
    print(f'Least squares fit: norm = {S:.4f} + {T:.4f} * log(n)')
    res_fit = S + T * log_n
    plt.plot(nlist, res_fit, label = 'Least Squares Fit')
    plt.xlabel('Matrix size n')
    plt.ylabel('Estimated Schur Multiplier Norm')
    plt.title('Operator Infinity Norm of Schur Multiplier with Adam Gradient Ascent')
    plt.legend()
    plt.grid(True)
    timestamp = time.strftime("%Y%m%d-%H%M%S")
    plt.savefig(f'plot4_n{nmax}_gaa_lse_{timestamp}.png')
    plt.show()
    print("Runtime: "+str((time.time()-start)/60)+" min")
```

B

Supporting Tables for Chapter 5

In Chapter 5, the results of the implementation shown in Appendix A are analyzed. In this analysis, the values obtained by certain parts of this code are used. The relevant data are shown in the following tables.

Value of p	**Converged at n=**	**Estimated norm**
1	25	2.0571
2	6	0.9999
3	18	1.0954
4	28	1.2940
5	13	1.2979
6	22	1.4878
7	22	1.5527
8	22	1.6042
9	25	1.6463
10	25	1.6855
11	25	1.7156
12	26	1.7430
13	25	1.7669
14	25	1.7917
15	25	1.8063
16	27	1.8227
17	25	1.8431
18	25	1.8506
19	25	1.8626
20	25	1.8734
21	25	1.8833
22	25	1.8924
23	25	1.9008
24	25	1.9085
25	25	1.9198

Table B.1: Supremum of Schur Multiplier Norm over $n \leq 30$ for different values of p.

Value of n	**Estimated norm**	**Fitted norm**	**Residual**
2	1.4138	1.4722	-0.0584
3	1.5712	1.5264	0.0448
4	1.6443	1.5648	0.0795
5	1.5221	1.5946	-0.0725
6	1.5869	1.6190	-0.0321
7	1.6069	1.6396	-0.0326
8	1.6847	1.6574	0.0273
9	1.6522	1.6731	-0.0210
10	1.7351	1.6872	0.0479
11	1.7775	1.7000	0.0776
12	1.7111	1.7116	-0.0005
13	1.7034	1.7223	-0.0188
14	1.7386	1.7322	0.0065
15	1.7707	1.7414	0.0293
16	1.7073	1.7500	-0.0427
17	1.8282	1.7581	0.0701
18	1.7421	1.7658	-0.0236
19	1.7628	1.7730	-0.0102
20	1.7572	1.7798	-0.0227
21	1.7837	1.7863	-0.0026
22	1.7548	1.7926	-0.0378
23	1.8026	1.7985	0.0041
24	1.7989	1.8042	-0.0053
25	1.8024	1.8096	-0.0072

Table B.2: Residuals of the Least Squares Fit for the Norm Estimation

C

Use of AI

With the purpose of being transparent, the use of Artificial Intelligence during the research and writing process of this thesis is discussed here. In particular, ChatGPT was used for the following purposes:

- Assisting in Python, mainly for debugging and optimizing the code;
- Improving the formal tone of the writing style in certain paragraphs;
- Correcting grammar and sentence structure;
- Implementing written proofs and generated data, such as the tables in Appendix B, in LaTeX format.

All AI-generated work was critically reviewed and adapted before being implemented in the thesis. AI was not used as a source itself, all statements are based on academic sources or independent reasoning. The content of the thesis is based on original analysis.