Fast Numerical Nonlinear Fourier Transform Algorithms for the Manakov Equation

L. de Vries

Master of Science Thesis

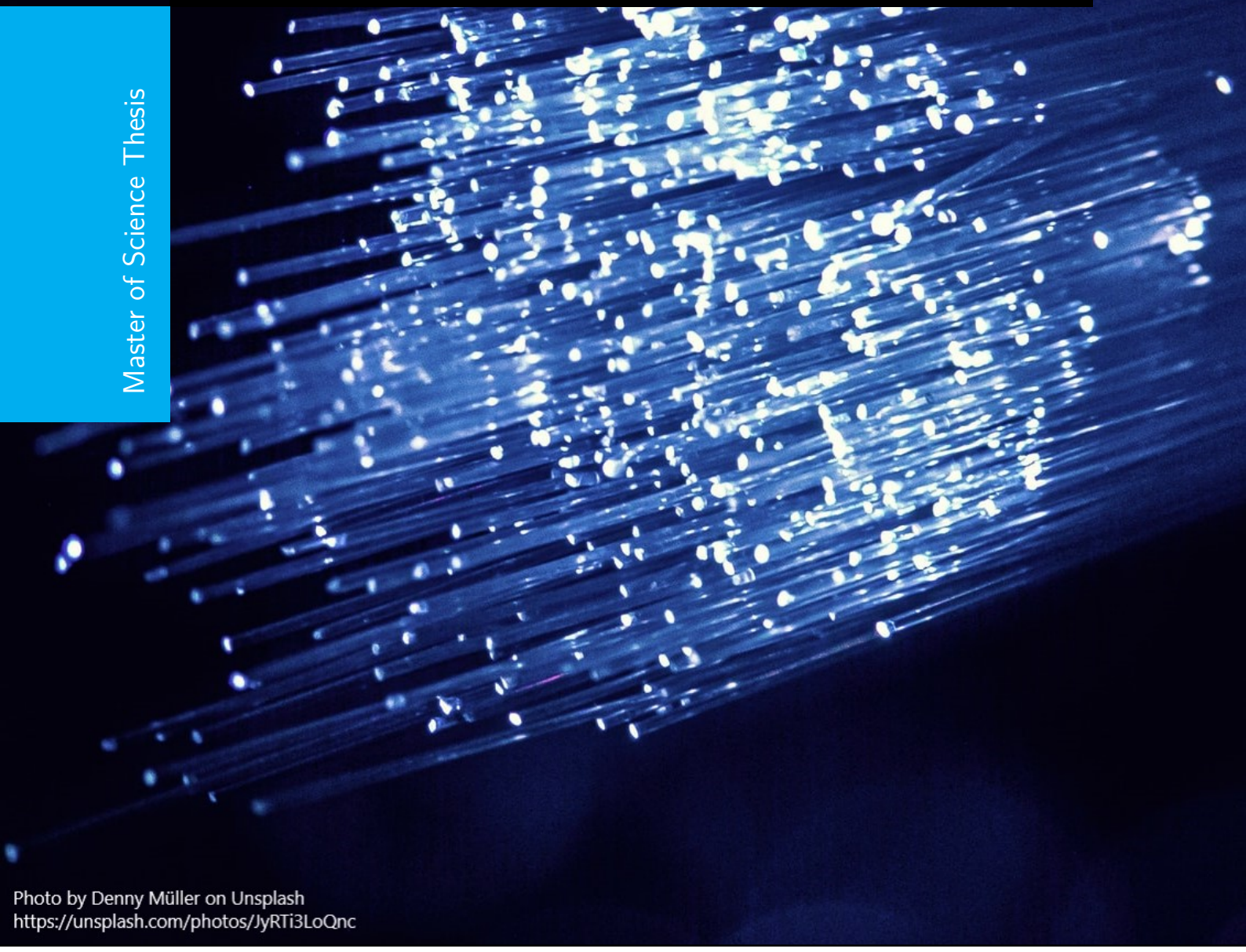


Photo by Denny Müller on Unsplash
https://unsplash.com/photos/JyRTi3LoQnc

TU Delft Delft University of Technology

Delft Center for Systems and Control

Fast Numerical Nonlinear Fourier Transform Algorithms for the Manakov Equation

Master of Science Thesis

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

L. de Vries

September 28, 2021

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

Abstract

Optical fibers form the backbone of our global data transmission infrastructure. As demands on global data transmission grow the capacity of these systems needs to be increased. The behaviour of light waves through these optical fibers is described by the Manakov Equation (ME), a system of nonlinear partial differential equations.

The ME is an integrable system, which can be solved analytically using Nonlinear Fourier Transforms. Recently, fiber-optic communication systems based on the Nonlinear Fourier Transform (NFT) of the ME have been proposed. Similar to the linear Fourier Transform, which decomposes a signal in linear frequency components, the NFT decomposes a signal in nonlinear frequency components. This nonlinear spectrum consists of a continuous and a discrete part. The continuous spectrum in general constitutes the whole real line. The discrete spectrum consists of distinct points in the complex plane which correspond to so-called solitons, which are stable wave forms. The evolution of the nonlinear spectrum along the fiber is trivial.

The nonlinear spectrum however cannot be computed analytically for most signals and therefore numerical methods are needed. The existing numerical methods have a high computational complexity of $O(D^2)$ for computing the continuous spectrum, with D the number of time samples of the signal. For the Nonlinear Schrödinger Equation (NSE), a simplification of the ME, more efficient numerical methods exist with a computational complexity of $O(D \log^2(D))$. In this thesis we present an extension of these so-called fast NFT methods to the ME. The resulting algorithms are second and fourth-order algorithms based on second and fourth order exponential integration methods respectively.

We developed open source software implementing the fast NFT algorithms for the ME and integrated them in the already existing Fast Nonlinear Fourier Transform (FNFT) software library. We provide detailed documentation and examples which allow other researchers to use the algorithms as tools or as a base for developing new algorithms. We furthermore test the accuracy of the developed algorithms against analytic examples. Of these examples, the rectangle signal and secant hyperbolic signal are new analytic examples for the ME to the best of our knowledge.

Table of Contents

List of Figures

Preface and acknowledgements

The topic of Nonlinear Fourier Transforms (NFTs) and fast NFT algorithms was first brought to my attention on the master thesis market organised by the Delft Center for Systems and Control (DCSC) department. It caught my interest as it seemed to be a way to dive deep into a mathematical topic a had not explored before. The end goal of the proposed project would be to contribute code to a C-based library. This meant I had to learn how to code in C, which I thought to be a nice challenge as well as a useful skill. Shrinivas Chimmalgi agreed to supervise my thesis on this topic and together with dr. ing. Sander Wahls the topic was further specified to fast NFTs for the Manakov Equation.

I am glad to have had Shrinivas as my daily supervisor and regret not seeing him more often in person because of the COVID-19 pandemic. I would like to thank him for all his support. I felt I could always ask him questions and he was very clear and patient when answering them. He always came up with some useful tips or tried to solve the problem together with me whenever I got stuck. Also the amount of help he provided while debugging my code was incredible. I would also like to thank Sander for his helpful feedback, mainly during the joint meetings together with Shrinivas and concerning the literature survey and final report. Together they made the thesis project a very educative and fun (but challenging) experience.

I would also like to thank my friends, who are much more important to me than they probably know. Thanks for the board games, the bike rides, the phone calls and the campfire nights. Thanks for listening and offering advice when I needed it. Your outsiders perspectives on my thesis project were invaluable. But mostly, I would like to thank them for all the type 1, 2 and 3 fun activities they joined me in over the years (link for those unfamiliar with the term: https://goeast.ems.com/three-types-of-fun/). They forced me to take my mind of the thesis work and helped me relax.

Lastly, I would like to thank my parents for their support during my thesis project and everything before that. They always supported me when I wanted to challenge myself, without pushing too hard. They gave me the time and space grow, but I knew and know they are always there to help me when I need it. Thank you.

Delft, University of Technology — L. de Vries
September 28, 2021

Chapter 1

Introduction

1-1 Introduction

We make video calls with friends far away, use streaming services to provide us with personalized entertainment and rely on cloud services to back up our files and access them at a moments notice. These examples show us how dependent we are on global data transmission systems, and we cannot deny the need for fast global data traffic in our society. Optical fiber communication systems nowadays form the backbone of our global data transmission infrastructure [3]. In optical fiber communication, information is carried by packets of light travelling through an optical fiber.

Figure 1-1 shows a schematic representation of such a communication system. The optical transmitter converts the electrical input signal to a light wave. It then passes through the communication channel, passing through optical amplifiers on the way. At the end of the channel the light waves are received and decoded by the optical receiver and the original signal is recovered. The performance of such a system is limited by multiple different effects in optical fibers [12]. The first of these effects is loss: the power of a signal deteriorates as the light wave propagates through the fiber. The optical amplifiers placed along the fiber counter this effect. These amplifiers however introduce their own noise which should be accounted for [12, Sec. 2.2]. The second effect is the dispersion: wave components with different frequencies travel at different speeds, which means that the signal is temporally broadened. Lastly, the Kerr nonlinearity plays a role. This means that the refraction index of the fiber is dependent on the intensity of the signal. The refraction index $r = c/v$ indicates how fast light travels through a medium, where c is the speed of light in a vacuum and v the speed of light through the specific medium. If we increase the power of the signal, which we might want to do to mitigate the effect of loss, this effect becomes more prominent [12, Sec. 2.2].

All these three effects on light waves travelling through an optical fiber are captured by the Manakov equation, as introduced by S.V. Manakov in 1974 [20]. The equation also accounts for *birefringence* of the fiber. In a birefringent material the refraction index depends on the polarization of the light waves travelling through it. This property is often present in optical

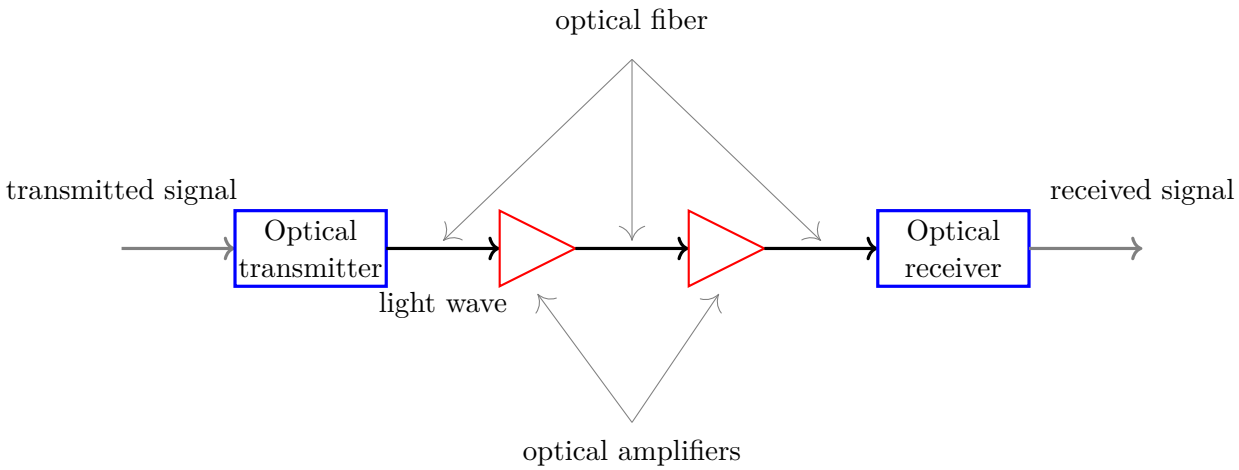


Figure 1-1: Schematic representation of an optical fiber communication system

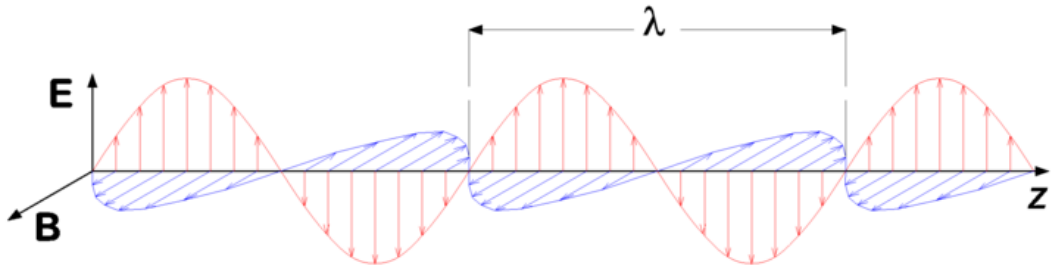


Figure 1-2: Schematic representation of polarization of light waves [43]

fibers [42]. The polarization of a transverse wave specifies the direction of the oscillation of this wave. This is visualized in Figure 1-2. The blue wave and the red wave both have the same amplitude, wave length, and velocity, but they oscillate in a different direction, albeit both perpendicular to the travelling direction. Therefore their polarization is different.

We give the normalized form of the Manakov equation here [20, Eq. 4]:

$$\begin{cases} jq_{1_x} + q_{1_{tt}} + 2\kappa(|q_1|^2 + |q_2|^2)q_1 &= 0 \\ jq_{2_x} + q_{2_{tt}} + 2\kappa(|q_1|^2 + |q_2|^2)q_2 &= 0 \end{cases} \tag{1-1}$$

where $\mathbf{q}(x,t) = \begin{bmatrix} q_1(x,t) & q_2(x,t) \end{bmatrix}^T$ is the *potential function*, the signal being sent through the fiber (the two elements of $\mathbf{q}(x,t)$ capture the polarizations). The constant j is the imaginary unit, $j^2 = -1$, and $\kappa = \pm 1$ is called the *dispersion constant* with -1 denoting the defocussing regime (normal dispersion) and $+1$ denoting the focussing regime (anomalous dispersion). Dispersion is the phenomenon where the velocity of a light wave depends on its wavelength. In the defocussing regime the velocity increases for increasing wavelength. In the focussing regime the velocity decreases for increasing wavelength. The subscripts x and t are used to indicate partial derivatives. The system of equations arises from the Maxwell equations when using cylindrical coordinates and boundary conditions belonging to an optical fiber [25]. After

some intermediate steps (a change of variables, normalization to go from equations involving physical units to dimensionless form, and replacing the space-varying envelope of the potential function $\mathbf{q}(x,t)$ by the average envelope over the fiber) for which we refer the reader to [19, Section 2], we arrive at Eq. (1-1).

As demands on global data transmission speed and volume keep rising, the current optical fiber communication systems are rapidly approaching their limits [35, Introduction]. As these systems are limited by among other things by the Kerr nonlinearity, incorporating this into models might help to better mitigate those effects. Recently a new class of optical communication techniques have been introduced which seek to exploit the nonlinearities of the fiber. The techniques falling under this new paradigm in the field of optical fiber communication are collected under the umbrella term Nonlinear Frequency Division Multiplexing (NFDM) [12, Sec. 4.1].

The downside of incorporating the nonlinearity of the fiber along with loss and dispersion effects is that the Manakov equation that governs these effects is a complicated nonlinear Partial Differential Equation (PDE). In general these type of equations lead to systems that cannot be solved analytically and one has to resort to (often computationally expensive) numerical approaches. However, while the evolution of $\mathbf{q}(x,t)$ described in the time domain is complicated, it turns out that the evolution described in the nonlinear frequency domain is quite simple [12, Chapter 3]. To take advantage of this, we need to use the so-called Nonlinear Fourier Transform (NFT) to get the nonlinear spectrum of the signal $\mathbf{q}(x,t)$ from its time-domain description.

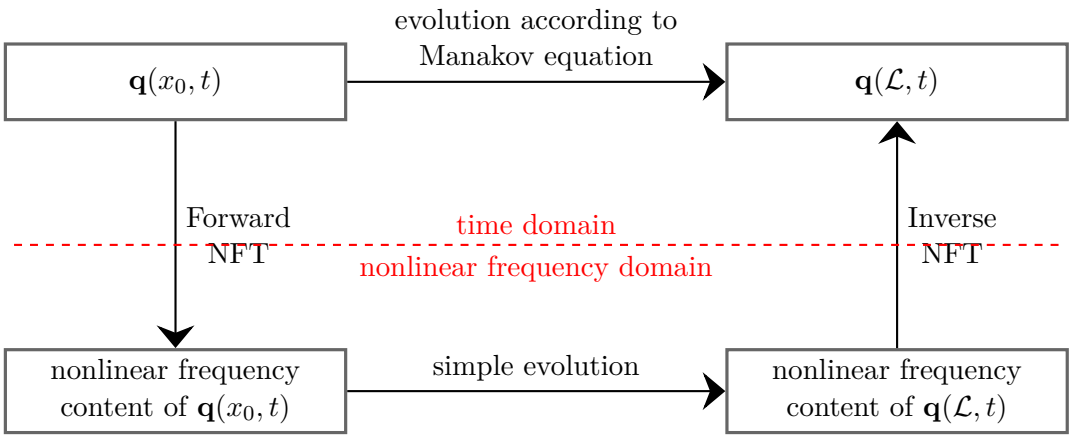


Figure 1-3: Schematic representation of the NFT method for solving PDEs

Figure 1-3 gives a schematic representation of this, where we understand $\mathbf{q}(x_0,t)$ to be the signal at the beginning of the fiber and $\mathbf{q}(\mathcal{L},t)$ the signal at the end of the fiber. We can draw parallels with the well-known Fourier Transform (FT) here. Where the FT can be used to decompose a signal in terms of linear frequency components (sinusoids), the NFT does something similar but decomposes the signal in nonlinear frequency components. The FT technique used to solve linear PDEs also has a nonlinear analogue using the NFT: it turns out that certain nonlinear PDEs, among which the Manakov equation, can be solved by moving to the nonlinear frequency domain where the evolution is simpler and going back to the time domain after that.

As noted earlier, a better understanding of the fiber nonlinearity through the Manakov equation and use of the NFT might help mitigate these effects. NFDM goes one step further. Instead of trying to minimize the nonlinear effect and using compensation techniques, NFDM exploits the nonlinearity and encodes information on the nonlinear spectrum. Figure 1-4 gives

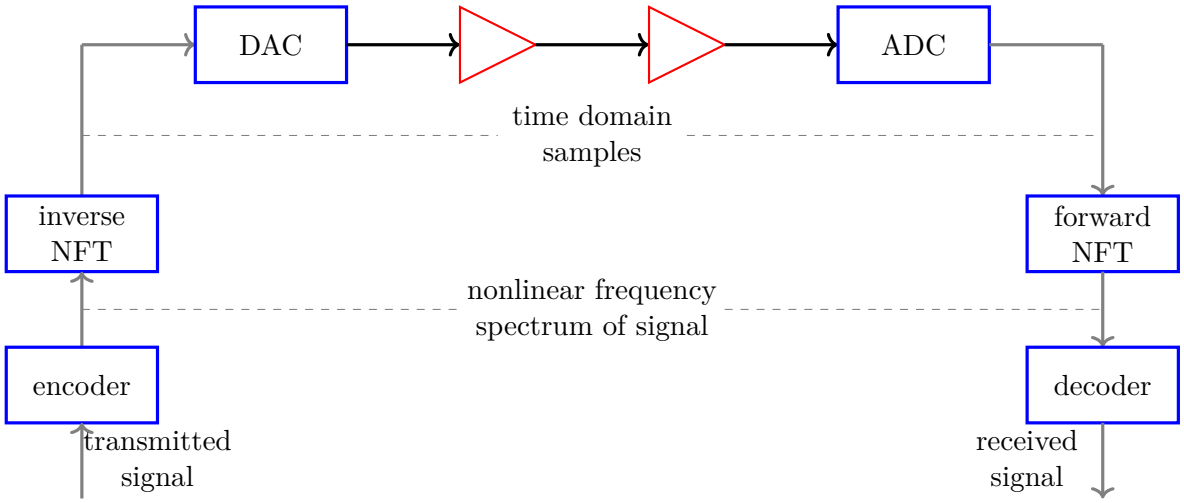


Figure 1-4: Schematic representation of Nonlinear Frequency Division Multiplexing

a schematic representation of such a communication system using NFDM. Before transmission we encode our message as a signal in the nonlinear frequency domain and use the inverse NFT to get samples of the signal $\mathbf{q}(x,t)$. The Digital-to-Analog converter (DAC) generates the optical signal from those samples. These steps together form the optical transmitter block from Figure 1-1. The transmission of the data through the optical fiber is the same as shown in Figure 1-1. After transmission of the signal we use an Analog-to-Digital Converter (ADC) and the forward NFT to get the nonlinear frequency content at the end of the fiber. Because the evolution of the nonlinear frequency content along the fiber is simple the decoder can easily recover the original signal. These last three steps form the optical receiver block from Figure 1-1. For more details on NFDM we refer to [12, Chapter 4].

Taking into account the polarization of the light waves offers an additional degree of freedom in which information can be encoded within NFDM, effectively doubling the information transmission rate through a channel with negligible performance degradation [11]. If the polarization is not accounted for, the behaviour of light waves in an optical fiber is given by the Nonlinear Schrödinger Equation (NSE). Its normalized form is given by [46]

$$jq_x + q_{tt} + 2\kappa|q|^2 q = 0. \tag{1-2}$$

We mention this because a lot of literature is available for NSE. The papers [44] and [45] provide the necessary background for optical fiber communication using the NFT in single polarization fibers, i.e. in fibers where the lightwave behaviour is governed by the NSE. The approach has been verified experimentally in [27] and [4] amongst others.

We turn our attention again to Figure 1-4 and Figure 1-3. In both of these scenario's we need to determine the nonlinear frequency spectrum from the time domain description of $\mathbf{q}(x,t)$. In other words, we need to take the forward NFT of $\mathbf{q}(x,t)$. To use these principles in the

optical fiber communication systems of the future we will need numerical procedures for this. We also need these methods to be fast: the lack of sufficiently fast and accurate algorithms is currently one of the main obstacles in the development of dual-polarization NFDM systems [14, Sec. VI], [11, Sec. 6]. A new group of forward NFT algorithms, called Fast Nonlinear Fourier Transforms (FNFTs), has been introduced for the NSE by Wahls and Poor [38], [40]. Furthermore, Wahls et. al. have been working on a C-based open source software library which makes these algorithms available for everyone to use easily [39]. In this thesis we aim to do the same for the Manakov equation. This brings us to the problem statement of this thesis:

In this thesis we aim to extend the first Fast Nonlinear Fourier Transform methods as proposed for the Nonlinear Schrödinger Equation to the dual-polarization case of the Manakov equation. We will develop these Fast Nonlinear Fourier Transform algorithms, integrate them in the existing open-source software library and benchmark them.

We hope that this will make fast NFT algorithms accessible to anyone working with NFT's and possibly contribute to further development of NFDM schemes and other optical communication techniques.

This thesis report is structured as follows. In Chapter 2 we provide the necessary mathematical foundation to determine the NFT of the Manakov Equation (ME). We also give the evolution of the nonlinear frequency components, explain how to use the NFT to solve PDE's and draw parallels with the linear FT. We also explain what the continuous and discrete spectra are. In Chapter 3 we give the steps to get a numerical approximation of the NFT. We start with the general procedure, then provide the expressions of the methods used as base methods for the fast forward NFT, introduce the general idea of the Fast Nonlinear Fourier Transform (FNFT) and end the chapter by citing some results for the NSE. In Chapter 4 we develop fast algorithms for the ME based on those for the NSE. Chapter 5 then explains the structure of the library, which methods we implemented and how they were implemented. In Chapter 6 we analyze the implemented methods by computing the NFT of various test signals with the different methods and comparing the numerical solution to the exact solution. We also compare the error-runtime trade-off of the different fast algorithms to each other. In Chapter 7 we of by summarize the results and contributions from this thesis work, draw some conclusions and give recommendations for future work.

1-2 Notation

In this section we provide a summary of the notations used in this report. Scalars and vectors are denoted by lowercase letters (e.g. a) and boldface lowercase letters (e.g. $\mathbf{a}$) respectively. Matrices and matrix operators are denoted by boldface uppercase letters (e.g. $\mathbf{A}$). To distinguish between continuous and discrete functions we will use round parentheses for continuous functions (e.g. $\mathbf{v}(t_n)$) and block parentheses for indexing a sampled function (e.g. $\mathbf{v}[n]$). Partial derivatives are denoted by subscripts (e.g. $q_t = \frac{\partial q}{\partial t}$. For the imaginary unit we use j; $j^2 = -1$. Complex conjugation is denoted by an asterisk * (e.g. a^*). For the computational complexity of an algorithm, we use uppercase O: $O(N^2)$. We use exp to denote exponentials, e.g. $\exp(a) = e^a$ and expm to denote matrix exponentials, e.g. $\text{expm}(\mathbf{A}) = e^{\mathbf{A}}$. The symbols $\mathbb{R}$, $\mathbb{C}$ and $\mathbb{N}$ are used to denote the set of real, imaginary and positive integer numbers respectively.

Notation	Definition
a	Scalar
$\mathbf{a}$	Vector
$\mathbf{A}$	Matrix
$\mathbf{v}(t_n)$	Function $\mathbf{v}(t)$ at time t_n
$\mathbf{v}[n]$	Sample of function $\mathbf{v}(t)$ at time t_n, index n
q_t	Partial derivative to t of $q(x,t)$
j	Imaginary unit, $j^2 = -1$
a^*	Complex conjugate of a
$\mathbf{q}_x$	Partial derivative to x of vector $\mathbf{q}$
expm	Matrix exponential
$O(\cdot)$	Computational order of complexity
$\exp(a)$	Exponent of a, e^a
$\text{expm}(\mathbf{A})$	Matrix exponential of $\mathbf{A}$, $e^{\mathbf{A}}$
$\mathbb{R}$	Set of real numbers
$\mathbb{C}$	Set of complex numbers
$\mathbb{N}$	Set of real positive integers

Table 1-1: Notations used in the report

Chapter 2

Mathematical background

The first Nonlinear Fourier Transform (NFT) (also known as the Inverse Scattering Transform (IST)) was introduced in 1967 by Gardner, Greene, Kruskal and Miura as a means of solving the Korteweg-de Vries (KdV) equation, a Partial Differential Equation (PDE) describing the behaviour of shallow-water waves [16], [17]. NFTs can be used to express a signal given in the time domain in nonlinear frequency domains. We can draw a parallel with the linear Fourier Transform (FT) here, which expresses data given in the time domain in the linear frequency domain. In both the linear and the nonlinear case, the evolution of the spectrum is simpler than the evolution of the time domain data. The spectrum also is complete in the sense that the time domain data can be recovered from it and it gives information about the frequency content of a signal that cannot easily be observed from the time domain expressions. At the time of publication of the paper [16] and [17] however, it was not yet clear if this method could be generalized to other PDEs. It was therefore a major breakthrough when Lax discovered the underlying framework in the KdV equation that made this method possible [18]. The KdV system is a so-called integrable system. We elaborate on this notion of integrability in Section 2-1. Shortly after publication of the paper by Lax, Zakharov and Shabat discovered that the Nonlinear Schrödinger Equation (NSE) is also an integrable system and applied the concept of the NFT to the NSE [46]. In his 1974 paper, Manakov both introduced the Manakov equation and outlined how to use the NFT to solve this equation. In the next section we introduce the concept of integrability and Lax pairs, which are operators related to a nonlinear PDE that play a key role in performing the NFT. After doing this in general terms we will give the expressions for the Manakov equation. The sections after that will introduce the NFT in a more precise mathematical manner.

2-1 Integrable systems and Lax pairs

Consider a system obeying a PDE of the form

$$\mathbf{q}_x = K(\mathbf{q}(x,t)). \tag{2-1}$$

Here and in the remainder of this thesis report x and t denote spatial and temporal variables respectively, the subscript denotes a partial derivative (e.g. $\mathbf{q}_x = \frac{\partial \mathbf{q}}{\partial x}$) and $K(\mathbf{q}(x,t))$ is a possibly nonlinear expression involving $\mathbf{q}(x,t)$ and its partial derivatives to t. The functions $\mathbf{q}(x,t)$ and $K(\mathbf{q}(x,t))$ can be vectors or scalars. The roles of x and t can be swapped depending on the system described by the PDE. This system is called an integrable system if there exists a Lax pair associated with the PDE:

Definition 1. *A pair of operators $\mathbf{L}(x)$ and $\mathbf{M}(x)$ is called a Lax pair of the PDE if they satisfy the Lax equation [44, Eq. 8]*

$$\frac{d\mathbf{L}}{dx} = [\mathbf{M}, \mathbf{L}] = \mathbf{ML} - \mathbf{LM}, \tag{2-2}$$

and this Lax equation evaluates to the PDE.

Both $\mathbf{M}$ and $\mathbf{L}$ are allowed to depend on $\mathbf{q}(x,t)$. There is no systematic method of finding these Lax pairs that works for every signal $\mathbf{q}(x,t)$. However, the Lax pairs of the equations mentioned in the introduction of this chapter are known. We refer to [17] or [12, Ex. 1] for the Lax pair of the Korteweg-de Vries equation and [12, Eq. 3.6, 3.7] for the Lax pair of the NSE. The Lax operators of the Manakov equation are given by [12, Eq. 3.15 - 3.16]

$$\mathbf{L}_{\text{Manakov}}(x) = j \begin{bmatrix} \frac{\partial}{\partial t} & -q_1(x,t) & -q_2(x,t) \\ -\kappa q_1(x,t)^* & -\frac{\partial}{\partial t} & 0 \\ -\kappa q_2(x,t)^* & 0 & -\frac{\partial}{\partial t} \end{bmatrix}, \tag{2-3}$$

$$\mathbf{M}_{\text{Manakov}}(x) = \begin{bmatrix} -2j\lambda^2 + j\kappa(|q_1|^2 - |q_2|^2) & 2\lambda q_1 + jq_{1_t} & +2\lambda q_2 + jq_{2_t} \\ -2\lambda\kappa q_1^* + j\kappa q_{1_t}^* & 2j\lambda^2 - j\kappa(|q_1|^2 - |q_2|^2) & 0 \\ -2\lambda\kappa q_2^* + j\kappa q_{2_t}^* & 0 & 2j\lambda^2 - j\kappa(|q_1|^2 - |q_2|^2) \end{bmatrix}. \tag{2-4}$$

The variables x and t have been omitted from the second operator to save space. An important property of the operator $\mathbf{L}(x)$ is that it is *isospectral* if $\mathbf{q}(x,t)$ solves the Manakov system [44, Lemma 1]. This means that even though the operator itself changes with x, its spectrum does not; the eigenvalues of $\mathbf{L}$ are independent of x. These constant eigenvalues are found by solving the eigenproblem

$$\mathbf{Lv} = \lambda \mathbf{v}, \tag{2-5}$$

where $\mathbf{v} = \mathbf{v}(\lambda, t)$ are the vector-valued eigenfunctions and λ is an eigenvalue of the operator $\mathbf{L}(x)$. We can rewrite Eq. (2-5) as a first order evolution equation

$$\mathbf{v}_t = \mathbf{Pv}. \tag{2-6}$$

In the case of the Manakov Equation (ME), we can derive the expression for $\mathbf{P}$ from Eq. (2-5)

by plugging in $\mathbf{L} = \mathbf{L}_{\text{Manakov}}$ in Eq. (2-5):

$$j \begin{bmatrix} \frac{\partial}{\partial t} & -q_1(x,t) & -q_2(x,t) \\ -\kappa q_1(x,t)^* & -\frac{\partial}{\partial t} & 0 \\ -\kappa q_2(x,t)^* & 0 & -\frac{\partial}{\partial t} \end{bmatrix} \mathbf{v} = \lambda \mathbf{v} \tag{2-7}$$

$$\Leftrightarrow \begin{cases} jv_{1t} - jq_1v_2 - jq_2v_3 & = \lambda v_1 \\ -j\kappa q_1^* v_1 - jv_{2t} & = \lambda v_2 \\ -j\kappa q_2^* v_1 - jv_{3t} & = \lambda v_3 \end{cases} \tag{2-8}$$

$$\Leftrightarrow \begin{cases} jv_{1t} & = \lambda v_1 + jq_1v_2 + jq_2v_3 \\ jv_{2t} & = -j\kappa q_1^* v_1 - \lambda v_2 \\ jv_{3t} & = -j\kappa q_2^* v_1 - \lambda v_3 \end{cases} \tag{2-9}$$

$$\Leftrightarrow \frac{\partial \mathbf{v}}{\partial t} = \underbrace{\begin{bmatrix} -j\lambda & q_1 & q_2 \\ -\kappa q_1^* & j\lambda & 0 \\ -\kappa q_2^* & 0 & j\lambda \end{bmatrix}}_{\mathbf{P}_{\text{Manakov}}} \mathbf{v}. \tag{2-10}$$

The system Eq. (2-10) is the vector version of the Zakharov Shabat (ZS) system, named after the authors of [46] who introduced it for the NSE. The version in Eq. (2-10) is central in the study of the NFT of the Manakov equation and we will refer to it as the Manakov Zakharov Shabat (MZS) problem.

2-2 The Nonlinear Fourier Transform

Suppose Eq. (2-1) is an integrable system and its Lax pair is known. Furthermore, impose two assumptions on $\mathbf{q}(x,t)$ [44, assumptions 1]:

Assumptions 1.

$$\begin{aligned} &\mathbf{q}(x,t) \to 0 \text{ as } |t| \to \infty, \\ &\mathbf{q}(t) \in L^1(\mathbb{R}). \end{aligned} \tag{2-11}$$

Here $L^1(\mathbb{R})$ is the space of real functions bounded in the p-norm with $p = 1$. The second assumption thus states that $||\mathbf{q}(x,t)||_1 = \int_{t=-\infty}^{t=+\infty} \mathbf{q}(x,t)dt < \infty$. These are reasonable assumptions in our case of optical fiber communication applications. They simply say that the magnitude of the signal goes to 0 asymptotically in time (first assumption), and that this happens sufficiently fast for the signal to have finite energy (second assumption).

We will now look at Eq. (2-1) in a slightly more abstract way. Consider $\mathbf{q}(x_0,t)$ to be an input to the system defined by the PDE. Now we wish to look at the "output" of the system, $\mathbf{q}(\mathcal{L},t)$, for some arbitrary but fixed $\mathcal{L} > x_0$. Figure 2-1 gives a schematic representation of this viewpoint. The PDE is completely defined by its Lax pair. The evolution of the system can thus be expressed either as the given PDE or as its Lax pair.

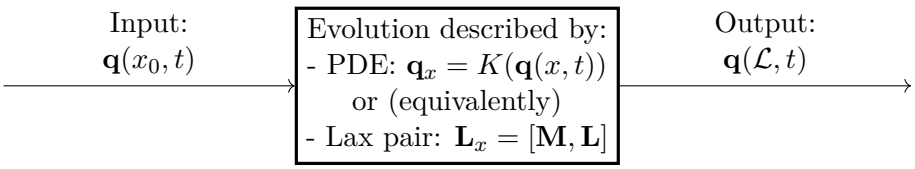


Figure 2-1: Schematic representation of the evolution of $\mathbf{q}(x,t)$ from its initial conditions

We will discuss the NFT analysis for the Manakov Equation. The Lax pair of the Manakov equation was given in Eq. (2-3) and Eq. (2-4), the corresponding $\mathbf{P}$ operator to form the ZS system for the Manakov equation was given in Eq. (2-10). Because the evolution of $\mathbf{q}(x,t)$ is completely defined by the PDE as well as the corresponding Lax pair, we can look at only the Lax pair. We start by analyzing the spectrum of the operator $\mathbf{L}(x)$: this is the nonlinear frequency content of the signal $\mathbf{q}(x,t)$ and is complete in the sense that $\mathbf{q}(x,t)$ can be completely recovered from it. As the eigenproblem Eq. (2-5) is equivalently written as Eq. (2-6), we choose to find the eigenfunctions $\mathbf{v}$ by analyzing Eq. (2-6). These eigenfunctions are functions of t and the eigenvalue λ (we omit the dependence on the fixed value $x = x_0$): $\mathbf{v}(t,\lambda)$. We are going to look at the behaviour at large absolute values of t and find the eigenfunctions spanning the eigenspace E_λ of $\mathbf{L}$. Due to the first assumption of Assumptions 1, Eq. (2-6) becomes

$$\mathbf{v}_t = \begin{bmatrix} -j\lambda & 0 & 0 \\ 0 & j\lambda & 0 \\ 0 & 0 & j\lambda \end{bmatrix} \mathbf{v} \quad \text{for } |t| \to \infty. \tag{2-12}$$

This is a simple decoupled system which has the general solution

$$\mathbf{v} = \begin{bmatrix} \alpha \exp(-j\lambda t) \\ \beta_1 \exp(j\lambda t) \\ \beta_2 \exp(j\lambda t) \end{bmatrix} \quad \text{for } |t| \to \infty. \tag{2-13}$$

For $\mathbf{v}(t, \lambda)$ to be a true eigenfunction of $\mathbf{L}$ the function both needs to be bounded and have finite energy for $|t| \to \infty$. We concentrate first on so-called generalized eigenfunctions, which only require $\mathbf{v}(t, \lambda)$ to be bounded, and elaborate more on the different eigenfunctions and their corresponding eigenvalues λ in Section 2-2-1. We try to find constants α, β_1, β_2 that lead to bounded solutions $\mathbf{v}(t, \lambda)$. We first turn our attention to the behaviour at the boundary $t \to \infty$ and consider all eigenvalues $\lambda \in \mathbb{C}^+$. In that case,

$$\mathbf{v} \to \begin{bmatrix} \alpha \exp(-j\lambda t) \\ 0 \\ 0 \end{bmatrix}. \tag{2-14}$$

The β_1, β_2 terms drop out because the combination of $\lambda \in \mathbb{C}^+$ and $t \to \infty$ yield an exponent with negative real part that goes asymptotically to 0. The exponent of the α term has positive real part in this case and blows up for $t \to \infty$; if we want the function $\mathbf{v}(t, \lambda)$ to be bounded α needs to be 0. In that case we can choose any value for β_1 and β_2. Therefore, two possible eigenfunctions are

$$\begin{bmatrix} 0 \\ \exp(j\lambda t) \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} 0 \\ 0 \\ \exp(j\lambda t) \end{bmatrix}, \tag{2-15}$$

and so are linear combinations of these vectors. As the vectors are also linearly independent, we have found two of the basis vectors for the eigenspace (the space of all eigenfunctions of $\mathbf{L}$) which we will call $\mathbf{\Psi}^P$:

$$\mathbf{\Psi}^P(t) \to \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \exp(j\lambda t) \quad \text{for } t \to \infty. \tag{2-16}$$

This condition uniquely determines the eigenfunctions. It is known that the eigenspace is symmetric in λ [44, Section IV B]: if λ is an eigenvalue, then so is λ^*. For these eigenvalues $\lambda \in \mathbb{C}^-$ we have

$$\mathbf{v} \to \begin{bmatrix} 0 \\ \beta_1 \exp(j\lambda t) \\ \beta_2 \exp(j\lambda t) \end{bmatrix} \text{ for } t \to \infty. \tag{2-17}$$

Here the α term drops out because the combination of $\lambda \in \mathbb{C}^-$ and $t \to \infty$ yields an exponent with negative real part that goes asymptotically to 0. Analogous to the $\lambda \in \mathbb{C}^+$ case, we now need β_1 and β_2 to be 0 for bounded $\mathbf{v}(t, \lambda)$ and we can choose any value for α. The basisvector for this eigenspace is thus

$$\bar{\mathbf{\Psi}}^P(t) \to \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \exp(-j\lambda t). \tag{2-18}$$

The same can be done for the boundary $t \to -\infty$ and this leads to the eigenfunctions

$$\boldsymbol{\Psi}^N(t) \to \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \exp(-j\lambda t) \quad \text{for } t \to -\infty, \tag{2-19}$$

$$\bar{\boldsymbol{\Psi}}^N(t) \to \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \exp(j\lambda t) \quad \text{for } t \to -\infty. \tag{2-20}$$

It is important to note here that the eigenfunctions evolve with time, while the eigenvalues stay the same. We are after all analyzing the spectrum of operator $\mathbf{L}$ which is an isospectral operator as we noted earlier. The equations for the eigenfunctions can be evaluated for all λ. Whether the chosen λ is an actual eigenvalue however depends on the properties of $\mathbf{v}(t, \lambda)$ for this value. We elaborate more in this in Section 2-2-1. We can now choose either the set of eigenfunctions on the boundary $t \to \infty$, $\boldsymbol{\Psi}^P$ and $\bar{\boldsymbol{\Psi}}^P$, or the set on the boundary $t \to -\infty$, $\boldsymbol{\Psi}^N$ and $\bar{\boldsymbol{\Psi}}^N$, as the independent basis of the eigenspace and express the other set in terms of this basis. We choose the eigenfunctions $\boldsymbol{\Psi}^P$, $\bar{\boldsymbol{\Psi}}^P$ as the basis:

$$\begin{aligned} \boldsymbol{\Psi}^N &= \bar{\boldsymbol{\Psi}}^P a(\lambda) + \boldsymbol{\Psi}^P b(\lambda), \\ \bar{\boldsymbol{\Psi}}^N &= \boldsymbol{\Psi}^P \bar{a}(\lambda) + \bar{\boldsymbol{\Psi}}^P \bar{b}(\lambda), \end{aligned} \tag{2-21}$$

with dimensions $a \in \mathbb{C}$, $\bar{a} \in \mathbb{C}^{2\times 2}$, $b \in \mathbb{C}^{2\times 1}$ and $\bar{b} \in \mathbb{C}^{1\times 2}$. One set of these coefficients, a and b or $\bar{a}$ and $\bar{b}$, is enough to describe the signal completely [10]. The coefficients $a(\lambda)$ and $b(\lambda)$ are called the Nonlinear Fourier coefficients. They are dependent on λ (and the chosen x_0), but not on t. We can thus choose the basisvectors at any t we wish and use them to calculate the nonlinear Fourier coefficients. Let us choose $t \to \infty$. The values of $\boldsymbol{\Psi}^P$ and $\bar{\boldsymbol{\Psi}}^P$ at this t are already known from the boundary conditions we imposed in Eq. (2-15). We fill them in for the first equation of Eq. (2-21):

$$\begin{aligned} \lim_{t\to\infty} \boldsymbol{\Psi}^N &= \lim_{t\to\infty} \left(\bar{\boldsymbol{\Psi}}^P a(\lambda) + \boldsymbol{\Psi}^P b(\lambda) \right) \\ &= \begin{bmatrix} a(\lambda) \\ 0 \\ 0 \end{bmatrix} \exp(-j\lambda t) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b_1(\lambda) \\ b_2(\lambda) \end{bmatrix} \exp(j\lambda t) \\ &= \begin{bmatrix} a(\lambda)\exp(-j\lambda t) \\ b_1(\lambda)\exp(j\lambda t) \\ b_2(\lambda)\exp(j\lambda t) \end{bmatrix} \end{aligned} \tag{2-22}$$

We can get $\boldsymbol{\Psi}^N$ at $t \to \infty$ by letting the boundary condition at $t \to -\infty$ Eq. (2-19) evolve in time according to Eq. (2-6). Then all eigenvectors appearing in Eq. (2-21) are known at the same time instant, namely $t \to \infty$. We can now get the NFT coefficients by multiplying both sides by $\exp(j\lambda t)$ or $\exp(-j\lambda t)$ for the a and $b_{1,2}$ coefficients respectively and selecting

the right entry of the results:

$$\begin{aligned}\lim_{t\to\infty} \mathbf{\Psi}^N \exp(j\lambda t) &= \begin{bmatrix} a(\lambda)\exp(-j\lambda t) \\ b_1(\lambda)\exp(j\lambda t) \\ b_2(\lambda)\exp(j\lambda t) \end{bmatrix} \exp(j\lambda t) \\ &= \begin{bmatrix} a(\lambda) \\ b_1(\lambda)\left(\exp(j\lambda t)\right)^2 \\ b_2(\lambda)\left(\exp(j\lambda t)\right)^2 \end{bmatrix}.\end{aligned} \tag{2-23}$$

and

$$\begin{aligned}\lim_{t\to\infty} \mathbf{\Psi}^N \exp(-j\lambda t) &= \begin{bmatrix} a(\lambda)\exp(-j\lambda t) \\ b_1(\lambda)\exp(j\lambda t) \\ b_2(\lambda)\exp(j\lambda t) \end{bmatrix} \exp(-j\lambda t) \\ &= \begin{bmatrix} a(\lambda)\left(\exp(-j\lambda t)\right)^2 \\ b_1(\lambda) \\ b_2(\lambda) \end{bmatrix}.\end{aligned} \tag{2-24}$$

The NFT coefficients are thus given by [12, eq. 3.24]

$$a(\lambda) = \lim_{t\to\infty} \mathbf{\Psi}_1^N \exp(j\lambda t), \quad b_i(\lambda) = \lim_{t\to\infty} \mathbf{\Psi}_{i+1}^N \exp(-j\lambda t). \tag{2-25}$$

So finding the NFT coefficients comes down to solving the MZS system Eq. (2-6). This is a scattering problem: in solving Eq. (2-6), we analyze how the eigenfunctions are scattered from $t \to -\infty$ to $t \to \infty$. The NFT coefficients are therefore sometimes called the scattering coefficients. They capture the scattering data of $\mathbf{q}(x_0, t)$ and can also be collected in the so-called scattering matrix $\mathbf{S}(x, \lambda)$. We can express Eq. (2-21) in terms of the scattering matrix as

$$\begin{bmatrix} \mathbf{\Psi}^N & \bar{\mathbf{\Psi}}^N \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{\Psi}}^P & \mathbf{\Psi}^P \end{bmatrix} \underbrace{\begin{bmatrix} a & \bar{b} \\ b & \bar{a} \end{bmatrix}}_{\mathbf{S}(x,\lambda)}. \tag{2-26}$$

2-2-1 Eigenvalues of the Manakov Zakharov-Shabat system

In the previous section we have found the eigenfunctions and nonlinear Fourier coefficients as functions of the eigenvalues λ. However, we have not yet explained how to find these λ's. We already mentioned that $\mathbf{v}(t, \lambda)$ can only be a (generalized) eigenvector if it stays bounded. From Eq. (2-21) it is clear that indeed bounded eigenfunctions are needed if we want meaningful (bounded) values for $a(\lambda)$ and $b(\lambda)$. For true eigenfunctions we also need the solution $\mathbf{v}(t, \lambda)$ to have finite energy.

The spectrum of $\mathbf{L}$ can consist of two parts: the continuous part and the discrete part. The continuous part in general constitutes the whole real line; if the imaginary part of λ is 0, we conclude from Eq. (2-13) that the entries of $\mathbf{v}(t, \lambda)$ oscillate for $|t| \to \infty$ and thus the norm of

this vector stays bounded. These solutions are the generalized eigenfunctions of $\mathbf{L}$. For the discrete part we consider only the upper half complex plane: the eigenspace is symmetric in the real axis, so if $\lambda \in \mathbb{C}^+$ is an eigenvalue we know $\lambda^* \in \mathbb{C}^-$ is an eigenvalue as well [44, Sec. IV B]. We look for eigenvalues such that $\mathbf{\Psi}^N$ in Eq. (2-21) stays bounded as $t \to \infty$. This is only possible if $a(\lambda) = 0$. We see this by filling out the values in Eq. (2-21):

$$\begin{aligned} \mathbf{\Psi}^N &= \bar{\mathbf{\Psi}}^P a(\lambda) + \mathbf{\Psi}^P b(\lambda) \\ &\to 0 + \begin{bmatrix} 0 \\ b_1 \exp(j\lambda t) \\ b_2 \exp(j\lambda t) \end{bmatrix}. \end{aligned} \tag{2-27}$$

Because $\lambda \in \mathbb{C}$ the exponent is negative and the second and third element of $\mathbf{\Psi}^N$ go to 0 exponentially, and $\mathbf{\Psi}^N$ thus stays bounded. Therefore the discrete part of the spectrum consists of all λ for which $a(\lambda) = 0$ [44, Sec. IV B]. These zeros of $a(\lambda)$ are isolated points, hence the name discrete spectrum. Only the focussing ME has discrete eigenvalues in its spectrum, the defocussing version does not [12, Proposition 4].

The discrete spectrum: soliton solutions

If a discrete spectrum exists, the potential function contains so-called solitons. A soliton is a stable waveform that propagates at a constant speed and retains its shape even after interacting with other solitons [46]. This is unlike the waves that can be found by analysis of linear PDE's, which are affected by interaction with other waves and which dissipate over time. Each discrete eigenvalue then corresponds to one of those solitons.
The potential function which gives rise to a single soliton is given in [20, Eq. 13]. The $b_{1,2}$ coefficients (and consequently the $\rho_{1,2}$ coefficients, see Eq. (2-28)) are 0 for this $\mathbf{q}(x,t)$ and we will therefor call it the single-soliton potential. In [34] the effects of multiple colliding solitons of the ME are treated and an expression for the multi-soliton potential function is given.

2-2-2 Definition of the Nonlinear Fourier Transform

With the information from previous the sections we are ready to state the definition of the NFT:

Definition 2. *The NFT of* $\mathbf{q}(t)$ *w.r.t. the Lax operator* $\mathbf{L}$ *is given by the two spectral functions*

$$\rho(\lambda) = \frac{\mathbf{b}(\lambda)}{a(\lambda)}, \quad \tilde{\rho}(\lambda_d) = \frac{\mathbf{b}(\lambda_d)}{a_\lambda(\lambda_d)}, \tag{2-28}$$

where $\lambda \in \mathbb{R}$ are the generalized eigenvalues which belong to the continuous spectrum, $\lambda_d \in \mathbb{C}$ for $d = 1, 2, ...N$ with N the number of eigenvalues are the eigenvalues of the discrete spectrum and $a_\lambda = \frac{da(\lambda)}{d\lambda}$.

Starting with an initial condition $\mathbf{q}(t) = \mathbf{q}(x_0, t)$, ρ and $\tilde{\rho}$ describe the scattering data of $\mathbf{q}(x_0, t)$. By evolving the eigenfunctions in space from $\mathbf{v}(x_0, \lambda)$ to $\mathbf{v}(\mathcal{L}, \lambda)$, we can derive the scattering data of $\mathbf{q}(\mathcal{L}, t)$. The evolution from $\mathbf{v}(x, \lambda)$ is given by

$$\mathbf{v_x} = \mathbf{M}\mathbf{v}. \tag{2-29}$$

This is clear from combining the eigenproblem Eq. (2-5) and the Lax equation Eq. (2-2):

$$\begin{aligned} \mathbf{L}\mathbf{v} &= \lambda\mathbf{v} \quad \text{(Derivative to } x\text{)} \\ \Leftrightarrow \mathbf{L_x}\mathbf{v} + \mathbf{L}\mathbf{v_x} &= \lambda\mathbf{v_x} \\ \Leftrightarrow (\mathbf{M}\mathbf{L} - \mathbf{L}\mathbf{M})\mathbf{v} + \mathbf{L}\mathbf{v_x} &= \lambda\mathbf{v_x} \\ \Leftrightarrow \lambda\mathbf{M}\mathbf{v} - \mathbf{L}\mathbf{M}\mathbf{v} &= \lambda\mathbf{v_x} - \mathbf{L}\mathbf{v_x} \\ \Leftrightarrow (\lambda I - \mathbf{L})\mathbf{M}\mathbf{v} &= (\lambda I - \mathbf{L})\mathbf{v_x} \Rightarrow \mathbf{v_x} = \mathbf{M}\mathbf{v} \end{aligned} \tag{2-30}$$

The spatial evolution of the NFT coefficients $a(\lambda)$ and $b_{1,2}(\lambda)$ is even simpler [15, Eq. 9a,9b]:

$$\begin{aligned} a(\lambda, x) &= a(\lambda, x_0) \\ b_i(\lambda, x) &= b_i(\lambda, x_0)\exp(-4i\lambda^2(x - x_0)). \end{aligned} \tag{2-31}$$

From the scattering data of $\mathbf{q}(\mathcal{L}, t)$, we can then derive the value of $\mathbf{q}(\mathcal{L}, t)$ by using the inverse NFT. These steps are schematically shown in Figure 2-2.

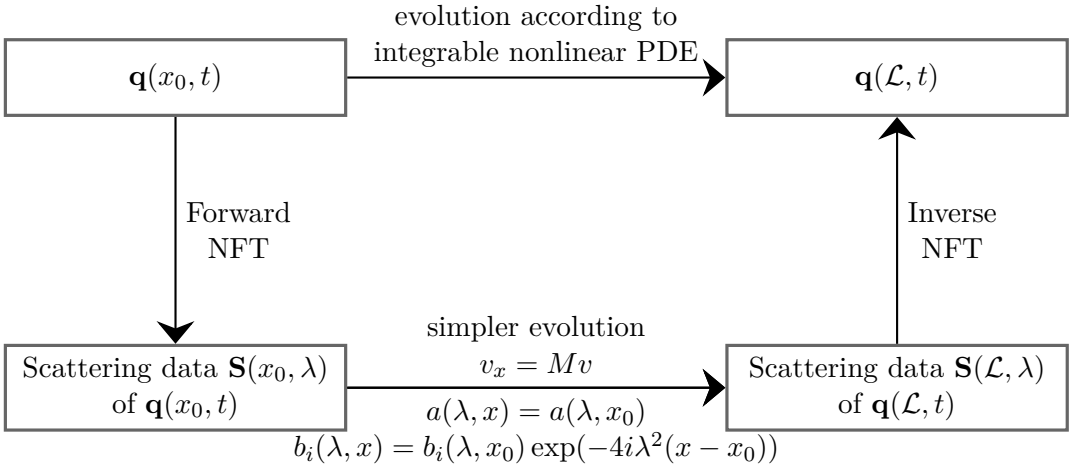


Figure 2-2: Schematic representation of the NFT method for solving PDEs

We see parallels with the linear FT here. If $\mathbf{p}_x = K(\mathbf{p}(x, t))$ is a linear PDE, we can use the linear FT to first express $\mathbf{p}(x, t)$ in the linear frequency domain where the evolution of $\mathbf{p}(x, t)$ is simpler and after that use the inverse FT to get the solution in the time domain. The NFT can be used in a similar manner to solve integrable nonlinear PDE's of the form Eq. (2-1). Similar to the linear FT, performing the forward NFT results in the nonlinear frequency content of the signal $\mathbf{q}(x, t)$. Just as with the linear spectrum, the nonlinear spectrum of a signal is informative in itself.

This thesis project focuses on the first step in Figure 2-2, performing the forward NFT to obtain the scattering data of $\mathbf{q}(x_0, t)$. There are only a few signals $\mathbf{q}(x_0, t)$ for which the

Nonlinear Fourier Spectrum is known analytically, and thus for general signals we resort to numerical procedures to compute the NFT. The next chapter introduces these numerical methods.

Chapter 3

Numerical methods for computing the Nonlinear Fourier Transform

In the first section of this chapter, we introduce the standard numerical methods for the Nonlinear Fourier Transform (NFT) that form the basis for the fast methods. Here and in the remainder of this thesis we call those standard methods the slow methods. In Section 3-2 we give the computational costs of these methods and make the notion of "slow" and "fast" methods precise. Finally, Section 3-3 introduces the Fast Nonlinear Fourier Transform (FNFT).

3-1 Slow Numerical Methods for the Forward NFT

The previous chapter introduced the notion of the NFT. In this thesis we focus on computing the forward NFT, i.e., determining the nonlinear Fourier coefficients $a(\lambda)$ and $b_{1,2}(\lambda)$ of initial data $\mathbf{q}(x_0, t)$. This is sufficient for finding the continuous spectrum of the signal and will be the main focus of this thesis. Finding the discrete spectrum involves finding the roots λ_d of $a(\lambda)$, numerically determining the derivative of $a(\lambda)$ and evaluating $b(\lambda_d)$ for all roots (see Eq. (2-28)). Especially evaluating $b(\lambda_d)$ can be difficult numerically as we see from Eq. (2-25) that the value in the limit might oscillate if λ has a nonzero real part. These steps offer their own challenges, and developing and comparing different approaches for the Manakov equation is an extensive topic in itself. We will however briefly discuss a simple method for finding the discrete eigenvalues in the context of fast methods in Section 4-8. The FNFT algorithm offers a benefit in this regard as well compared to the slow methods that we would like to highlight, and I also implemented this method in the library for the Manakov Equation (ME).

For finding the continuous spectrum, the most important numerical step is to evolve the eigenvectors from their initial conditions according to Eq. (2-6) to get the eigenvectors spanning the solution space. Solving this Manakov Zakharov Shabat (MZS) system is nothing more than numerically solving an Ordinary Differential Equation (ODE). We restrict ourselves to samples of $\mathbf{q}(x, t)$ on the interval $t \in [T_1, T_2]$. As long as we choose this interval sufficiently

large the truncation error is small because $\mathbf{q}(x,t)$ tends to 0 outside this interval by the first assumption in Assumptions 1 from Section 2-2. We divide this interval into $D-1$ subintervals of length h, so $D-1 = \frac{T_2-T_1}{h}$. Assume the potential function $q(x,t)$ is sampled at the gridpoints, and $\mathbf{v}(t)$ in the middle of these intervals and at $T_2 + \frac{1}{2}h$:

$$\begin{aligned}
\mathbf{q}[1] &= \mathbf{q}(x, T_1), \\
\mathbf{q}[2] &= \mathbf{q}(x, T_1 + h), \\
&\vdots \\
\mathbf{q}[n] &= \mathbf{q}(x, T_1 + (n-1)h), \\
&\vdots \\
\mathbf{q}[D] &= \mathbf{q}(x, T_2),
\end{aligned} \tag{3-1}$$

$$\begin{aligned}
\mathbf{v}[0] &= \mathbf{v}(\lambda, T_1 - \frac{1}{2}h), \\
\mathbf{v}[1] &= \mathbf{v}(\lambda, T_1 + \frac{1}{2}h), \\
\mathbf{v}[2] &= \mathbf{v}(\lambda, T_1 + 1\frac{1}{2}h), \\
&\vdots \\
\mathbf{v}[n] &= \mathbf{v}(\lambda, T_1 + (n - \frac{1}{2})h), \\
&\vdots \\
\mathbf{v}[D] &= \mathbf{v}(\lambda, T_2 + \frac{1}{2}h).
\end{aligned} \tag{3-2}$$

We would like to stress that the time instances for the samples of $\mathbf{v}(t)$ and $\mathbf{q}(t)$ are not the same: there is a half timestep difference between them. This is because most methods we introduce in Section 3-1-1 require samples of $\mathbf{q}(x,t)$ halfway between samples of $\mathbf{v}(t)$. We will use t_n to refer to the time instances where the samples of $\mathbf{q}(t)$ are taken: $\mathbf{q}[n] = \mathbf{q}(t_n)$. For the time instances of $\mathbf{v}(t)$ we have $\mathbf{v}[n] = \mathbf{v}(t_{n+1/2})$

For solving ODE's many numerical integration schemes are available. In this survey we will restrict ourselves to one-step schemes with equispaced samples as they are the basis for the FNFT procedure. In a one-step scheme the following iteration scheme is used to propagate $\mathbf{v}(\lambda,t)$ from its initial conditions:

$$\mathbf{v}[n,\lambda] = \mathbf{\Phi}[n,\lambda]\mathbf{v}[n-1,\lambda]. \tag{3-3}$$

The matrix $\mathbf{\Phi}$ is called the *transition matrix*. It is specific to the chosen numerical integration method and dependent on the sample $\mathbf{q}[n]$ (and possibly on the samples surrounding $\mathbf{q}[n]$). Once we have propagated $\mathbf{v}(\lambda,t)$ from the initial condition in Eq. (2-19) $\mathbf{\Psi^N} = \lim_{t\to-\infty}\mathbf{v}(t) \approx \mathbf{v}(T_1 - \frac{1}{2}h) = \mathbf{v}[0]$ to get an approximation of $\lim_{t\to\infty}\mathbf{v}(\lambda,t)$, we get the numerical approximations for the NFT coefficients in Eq. (2-25) from

$$\begin{aligned}
\hat{a}(\lambda) &= \mathbf{v}_1(T_2 + \frac{1}{2}h, \lambda)\exp\left(j\lambda(T_2 + \frac{1}{2}h)\right) = \mathbf{v}_1[D]\exp\left(j\lambda(T_2 + \frac{1}{2}h)\right), \\
\hat{\mathbf{b}_\mathbf{i}}(\lambda) &= \mathbf{v}_{i+1}(T_2 + \frac{1}{2}h, \lambda)\exp\left(-j\lambda(T_2 + \frac{1}{2}h)\right) = \mathbf{v}_{i+1}[D]\exp\left(-j\lambda(T_2 + \frac{1}{2}h)\right).
\end{aligned} \tag{3-4}$$

We should keep in mind here that $a(\lambda)$ and $\mathbf{b}(\lambda)$ are functions of the spectral parameter λ. When carrying out numerical computations using a slow method however, we do not get these coefficients as functions: we have to choose a specific value $\lambda = \xi$ and do all computations for this value, leading to $a(\xi)$ and $\mathbf{b}(\xi)$: for each λ we wish to know the NFT coefficients for, we have to carry out the computation again.

3-1-1 Expressions for the slow numerical methods

In this section we look at four slow methods: BO (Boffetta-Osborne, [7]), $\mathrm{CF}_2^{[4]}$ (Commutator-Free method, [9]), TES4 (Triple Exponential Scheme, [24]) and RK4 (fourth-order Runge-Kutta). The first three form the basis for the fast methods as implemented in the software library for the Manakov equation. These are exponential methods. We elaborate more on those in Section 4-1 but one of their properties that their transition matrices consist of (products of) exponential matrices which is key to the FNFT algorithm later on in Chapter 4. RK4 has been used in [45] and [8] in higher-order (that is, higher than second order error decay) NFT algorithms for the Nonlinear Schrödinger Equation (NSE). Although none of the methods implemented in the library was based on RK4, we include it to serve as a reference point for readers less familiar with numerical integration schemes as it is quite well-known. All methods discussed in this section are one-step methods.
We note once more that any numerical integration scheme could be used to solve the MZS system. Notably, the Crank-Nicholson scheme [45, Sec. III D] and the Ablowitz-Ladik discretization [45, Sec. III E] have been used in the past for the original Zakharov Shabat (ZS) problem. The authors of [36] compare those two methods along with BO and RK4 in the context of the ZS system. Crank-Nicholson and Ablowitz-Ladik offered a worse trade-off between number of samples and error in all their testcases, and as they do not form the basis of any of the fast methods we developed for the ME either we chose not to include them in our further discussion.

BO

Boffetta and Osborne applied this method to the NSE in their 1991 article [7]. The idea behind the method is simple: we assume that the matrix $\mathbf{P}$ is constant on the interval $(t_n - h/2, t_n + h/2)$. We call this constant matrix $\mathbf{P}_n$. Then $\mathbf{P}$ becomes a piecewise constant matrix of which the solution is known:

$$\mathbf{v}_t(t) = \mathbf{P}_n \mathbf{v}(t) \qquad \text{for} \quad t_{n-1/2} < t < t_{n+1/2} \tag{3-5}$$

$$\Rightarrow \mathbf{v}(t) = \operatorname{expm}(\mathbf{P}_n(t - t_{n-1/2}))\mathbf{v}(t_{n-1/2}) \qquad \text{for} \quad t_{n-1/2} < t < t_{n+1/2}. \tag{3-6}$$

where expm is a matrix exponential. So this gives us

$$\mathbf{v}[n] = \underbrace{\operatorname{expm}\left(\mathbf{P}[n]h\right)}_{\mathbf{\Phi}_{BO}} \mathbf{v}[n-1], \tag{3-7}$$

where we have

$$\mathbf{P}[n] = \begin{bmatrix} -j\lambda & q_1[n] & q_2[n] \\ -\kappa q_1^*[n] & j\lambda & 0 \\ -\kappa q_2^*[n] & 0 & j\lambda \end{bmatrix}. \tag{3-8}$$

This method is a second order method [7, Sec. 4]. The expression given in Eq. (3-7) has a third order local truncation error. That means that the error for each time step is $\mathbf{v}[n+1]_{\text{exact}} - \mathbf{v}[n+1]_{\text{BO}} = \mathcal{O}(h^3)$ if we assume that for time step n there are no errors: $\mathbf{v}[n]_{\text{BO}} = \mathbf{v}[n]_{\text{exact}}$. If we perform D numerical integration steps with the BO method, the total error will be $D\mathcal{O}(h^3)$. This is called the global truncation error. Over the complete interval $t \in [T_1 - \frac{1}{2}h, T_2 + \frac{1}{2}h]$ we have $h = \frac{(T_2+\frac{1}{2}h)-(T_1-\frac{1}{2}h)}{D}$ and thus $D \sim \frac{1}{h}$. The global truncation error is thus $\mathcal{O}(\frac{1}{h}h^3) = \mathcal{O}(h^2)$. BO therefore is a second order method; if we halve the timestep size in the BO method, the error will be $\frac{1}{2}^2 = \frac{1}{4}$ of the original error as long as other types of errors do not dominate. If a relatively low error is desired, a high number of samples might be needed. If the error can be a bit higher however this method might be a good choice as the computational complexity is fairly low and it is faster than most higher order methods for the same number of samples.

$\mathsf{CF}_2^{[4]}$

This method is similar to the BO method, but instead of assuming $\mathbf{P}(t)$ to be constant over each interval it uses more samples to more accurately represent the matrix exponential. It is based on the fourth-order commutator free method CF4 in [33] and was put forward as a numerical method for the NFT of the NSE in [9]. We label this method as $\text{CF}_2^{[4]}$. The subscript indicates the number of matrix exponentials and the superscript gives the order of the method. For the Manakov equation the expression for the transition matrix is [10, Eq. 22]

$$
\begin{aligned}
\mathbf{\Phi}[n] &= \operatorname{expm}(h(a_2 A_1 + a_1 A2))\operatorname{expm}(h(a_1 A_1 + a_2 A2)),\\
a_1 &= \frac{1}{4} + \frac{\sqrt{3}}{6}, \quad a_2 = \frac{1}{4} - \frac{\sqrt{3}}{6},\\
A_1 &= \begin{bmatrix} -j\lambda & q_1(t_n - \frac{\sqrt{3}}{6}h) & q_2(t_n - \frac{\sqrt{3}}{6}h) \\ -q_1^*(t_n - \frac{\sqrt{3}}{6}h) & j\lambda & 0 \\ -q_2^*(t_n - \frac{\sqrt{3}}{6}h) & 0 & j\lambda \end{bmatrix},\\
A_2 &= \begin{bmatrix} -j\lambda & q_1(t_n + \frac{\sqrt{3}}{6}h) & q_2(t_n + \frac{\sqrt{3}}{6}h) \\ -q_1^*(t_n + \frac{\sqrt{3}}{6}h) & j\lambda & 0 \\ -q_2^*(t_n + \frac{\sqrt{3}}{6}h) & 0 & j\lambda \end{bmatrix}.
\end{aligned}
\tag{3-9}
$$

We should notice that this method requires different samples than those taken at the grid-points. Also, the required samples are not equidistant, so simply redefining $\mathbf{v}[n]$ and shifting at which points this vector is calculated will not work. Sampling the potential function at non-equidistant time instances might be impractical or even impossible and thus interpolation techniques are used to acquire the samples. This should not be a problem however if the signal is bandlimited. In [9] local cubic interpolation is used for these samples. The FNFT library uses bandlimited Fourier interpolation. This is a fourth-order method, and thus more suitable if a lower error is desired.

TES4

The third method of interest was introduced in [24]. The authors set out to develop a one-step scheme to solve ODE's. They derived a condition on the expansion of the transition matrix $\mathbf{\Phi}$ such that the resulting method is fourth order. This expansion should have the form given in [24, Eq. 33]. In the symmetrical case, that is, if the points to evaluate $\mathbf{v}(t)$ are midway between the samples of $\mathbf{q}(t)$, it turns out that the series expansion of the exponential in Eq. (3-10) matches the form from [24, Eq. 33] exactly up until the fourth order term. Therefore, using this matrix exponential as a transition matrix in a one-step scheme will yield a fourth order method:

$$\mathbf{\Phi}_{TES4} = \text{expm}\left(\frac{h^2}{12}\mathbf{P}^{(1)} + \frac{h^3}{48}\mathbf{P}^{(2)}\right)\text{expm}(h\mathbf{P})\text{expm}\left(\frac{-h^2}{12}\mathbf{P}^{(1)} + \frac{h^3}{48}\mathbf{P}^{(2)}\right). \tag{3-10}$$

Matrices $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ are numerical approximations of the element-wise first and second derivatives of the matrix $\mathbf{P}$. The authors of [24] used finite differences for those numerical approximations. For each element of $\mathbf{q}(t)$ the finite difference approximations are

$$q_i^{(1)}[n] = \frac{q_i[n+1] - q_i[n-1]}{2h}, \tag{3-11}$$

$$q_i^{(2)}[n] = \frac{q_i[n-1] - 2q_i[n] + q_i[n+1]}{h^2}. \tag{3-12}$$

For matrices, we apply these expressions element-wise. We note that the entries involving λ do not change for different indices n and they cancel. Therefor we get

$$\mathbf{P}^{(1)}[n] = \begin{bmatrix} 0 & q_1[n]^{(1)} & q_2[n]^{(1)} \\ -\kappa q_1^*[n]^{(1)} & 0 & 0 \\ -\kappa q_2^*[n]^{(1)} & 0 & 0 \end{bmatrix} \tag{3-13}$$

$$\mathbf{P}^{(2)}[n] = \begin{bmatrix} 0 & q_1[n]^{(2)} & q_2[n]^{(2)} \\ -\kappa q_1^*[n]^{(2)} & 0 & 0 \\ -\kappa q_2^*[n]^{(2)} & 0 & 0 \end{bmatrix} \tag{3-14}$$

This scheme uses one more matrix exponential than the $\text{CF}_2^{[4]}$ method. However, as the first and third matrix exponential are independent of λ, the computation times for TES4 will be lower than those for $\text{CF}_2^{[4]}$ for identical numbers of samples.

RK4

The idea behind fourth-order Runge-Kutta (RK4) is to compute approximations k_1, k_2, k_3, k_4 of $\mathbf{v}(x, t_{n-1/2} + \tau)$ for multiple values of τ where $0 \leq \tau \leq h$ and combine them such that the result is a higher-order approximation of $\mathbf{v}(x, t_{n+1/2})$. The approximations k_i are dependent on the previous approximations k_j with $j < i$. [37, Sec. 6.5] gives the expressions for the

RK4 method:

$$\begin{aligned}
\mathbf{v}[n+1] &= \mathbf{v}[n] + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\
k_1 &= h\mathbf{P}[n+\frac{1}{2}]\mathbf{v}[n], \\
k_2 &= h\mathbf{P}[n+1](\mathbf{v}[n] + \frac{1}{2}k_1), \\
k_3 &= h\mathbf{P}[n+1](\mathbf{v}[n] + \frac{1}{2}k_2), \\
k_4 &= h\mathbf{P}[n+1\frac{1}{2}](\mathbf{v}[n] + k_3).
\end{aligned} \tag{3-15}$$

We refer to [41] for more details. In Eq. (3-15), $\mathbf{P}[n+\frac{1}{2}]$ is the matrix $\mathbf{P}$ with values for $\mathbf{q}(t)$ sampled at points $t_{n+\frac{1}{2}}$. These samples can either be attained by interpolation or one can choose to determine $\mathbf{v}(t)$ at only half the discretization points. The method can be written in the form Eq. (3-3) by plugging the expression for k_3 in the expression for k_4 etc. until only $\mathbf{v}[n]$ and $\mathbf{P}$ are present. Rearranging yields the form Eq. (3-3).

3-2 Computational complexity of the slow methods

In Algorithm 1 we give the pseudocode for computing the continuous spectrum using a slow numerical NFT method:

Algorithm 1 Computing the NFT using a slow method

Input: $\mathbf{q}[1], ..., \mathbf{q}[D]$, T_1, T_2, desired λ-interval, M
Output: $a[1], ...a[M]$, $\mathbf{b}[1], ...\mathbf{b}[M]$
Discretize λ-interval with M points
for i=1:M **do**
  for n=1:D **do**
    $\Phi = f(P(\lambda[i], t_n))$
    $\mathbf{v}[n+1] = \Phi\mathbf{v}[n]$
  end for
  $a[i] = \mathbf{v}_1[D] \exp\left(j\lambda[i](T_2 + \frac{1}{2}h)\right)$
  $\mathbf{b_k}[i] = \mathbf{v}_{k+1}[D] \exp\left(-j\lambda[i](T_2 + \frac{1}{2}h)\right)$
end for

This pseudocode clearly shows why the slow methods might have high computation times: all calculations have to be repeated for all M values of λ. This is the key property of all slow methods. Depending on how large the spectral parameter domain is that we are interested in, the computation time can add up quickly. We assume that the computational cost for the inner loop is dominated by the cost of one matrix-vector product. The cost of computing one or more matrix exponentials to get the transition matrix $\Phi = f(P(\lambda[i], t_n))$ is dependent on the chosen method and might be even higher than the cost of the matrix-vector product, so wit this assumption we are determining a lower bound for the computational cost. For a 3×3 matrix and a 3×1 vector we have 9 multiplications and 6 additions and thus a total of $15D$

floating point operations (FLOPs) if we carry out the loop D times. Taking into account the outer loop the total computational costs are $15DM$ FLOPs, which is $O(MD)$. The evaluation of the NFT coeffcients will add $O(M)$ FLOPs so the total computational cost stays $O(MD)$. It is common to choose the same amount of samples in the frequency domain as in in the time domain, $M = D$, so the computational costs for computing the continuous spectrum are $O(D^2)$. Fast Nonlinear Fourier Transform methods are methods whose computational cost is lower than $O(D^2)$. This does not mean that their runtimes are always lower than the runtime of a slow methods: *it just means that the runtime as a function of samples grows less fast for fast methods.* For lower numbers of samples, the actual runtime of a slow method might be lower. Where exactly the crossover point lies depends on the specific methods, implementations and computer architecture.

3-3 The Fast Nonlinear Fourier Transform

Section 3-1 outlined how to numerically determine the NFT of an integrable system. In Section 3-2 we highlighted why the slow methods can become relatively slow for high numbers of samples. The development of faster numerical NFT methods is essential to study new optical fiber communication techniques such as Nonlinear Frequency Division Multiplexing (NFDM) and apply them in practice. In 2013, Wahls and Poor introduced the so-called Fast Nonlinear Fourier Transform as a faster alternative to the slow methods and applied it to the NSE [38].

They present a general approach which, starting from D samples of the potential $\mathbf{q}(t)$, computes a numerical approximation of the continuous spectrum in $O(D\log^2 D)$ FLOPs and determines a numerical approximation of the discrete spectrum in $O(D^2)$ FLOPs. The idea is to use polynomial matrices as the transition matrices in Eq. (3-3). To understand the advantage of polynomial transition matrices, we need to write $\mathbf{v}[D,\lambda]$, which is an approximation of $\mathbf{v}(T_2+\frac{1}{2}h,\lambda)$, as a function of all transition matrices $\mathbf{\Phi}[n,\lambda]$ and the initial condition $\mathbf{v}[0,\lambda]=\mathbf{v}(T_1-\frac{1}{2}h,\lambda)$:

$$\begin{aligned}
\mathbf{v}[1,\lambda] &= \mathbf{\Phi}[1,\lambda]\mathbf{v}[0,\lambda],\\
\mathbf{v}[2,\lambda] &= \mathbf{\Phi}[2,\lambda]\mathbf{v}[1,\lambda] = \mathbf{\Phi}[2,\lambda]\mathbf{\Phi}[1,\lambda]\mathbf{v}[0,\lambda],\\
\mathbf{v}[3,\lambda] &= \mathbf{\Phi}[3,\lambda]\mathbf{v}[2,\lambda] = \mathbf{\Phi}[3,\lambda]\mathbf{\Phi}[2,\lambda]\mathbf{\Phi}[1,\lambda]\mathbf{v}[0,\lambda],\\
&\vdots\\
\mathbf{v}[D,\lambda] &= \left(\prod_{i=1}^{N}\mathbf{\Phi}[i,\lambda]\right)\mathbf{v}[0,\lambda].
\end{aligned} \tag{3-16}$$

Instead of multiplying the vector $\mathbf{v}[n,\lambda]$ with the transition matrix for each timestep we first multiply all polynomial transition matrices, which yields the total transition matrix $\prod_{i=1}^{N}\mathbf{\Phi}[i,\lambda]$ as a polynomial in λ. We then multiply the initial condition $\mathbf{v}[1,\lambda]$ directly with the total transition matrix to get $\mathbf{v}[D]$. Note that the total transition matrix is a function of λ. It thus needs to be calculated only once and can then be evaluated for all desired λ. This is different from the slow methods, which evaluate the transition matrix for all λ's individually. Moreover, because $\mathbf{\Phi}[n]$ is a polynomial matrix in λ for all n, the product of these matrices can be computed efficiently by multiplying the matrices in a tree-wise manner and using fast polynomial multiplication [38, Sec. V]. If $\mathbf{\Phi}[n,\lambda]$ is not a polynomial matrix a coordinate transform $z=f(\lambda)$ can be used to arrive at a transition matrix that is a polynomial in z. The authors of [10] expand upon this idea by using exponential one-step methods as the starting point, and use a combination of an exponential splitting from [28] and a coordinate transform $z=f(\lambda)$ to both to arrive at a transition matrix that is a polynomial in z and map z onto the unit circle in the complex plane. The polynomial form of the transition matrix allowed them to use the ideas put forward in [38]. The mapping onto the unit circle prevents some numerical issues as the norm of z^p is 1 for each p and this norm would either grow or decay very fast if z were not on the unit circle. It also allows for the use of the chirp-Z transform as a fast polynomial evaluation algorithm. In Chapter 4 we elaborate on all the steps and summarize the fast NFT approach for the Manakov system in Section 4-11. The next section cites some results of applying fast NFT algorithms to the NSE.

3-3-1 Results of the fast NFT for the NSE

The FNFT approach outlined in the previous subsection has been successfully applied to the NSE. In [10], the authors use both a fast implementation of BO (which they call $\mathrm{CF}_1^{[2]}$, or $\mathrm{FCF}_1^{[2]}$ for the fast version) and of the $\mathrm{CF}_2^{[4]}$ method ($\mathrm{FCF}_2^{[4]}$). In this article the authors used bandlimited Fourier interpolation for (F)$\mathrm{CF}_2^{[4]}$ instead of the cubic spline interpolation proposed in [9]. Both methods required a significantly lower execution time to achieve the same error compared to the slow implementations as outlined in Section 3-1-1. The tests were done using test signals for which the NFTs are known analytically.

In [22] a fast implementation of the TES4 method (FTES4) is presented and compared to the slow implementation. The fast method is about an order of magnitude faster than the slow counterpart when the number of samples in the frequency domain is kept constant at $M = 1025$. However, the error is also an order of magnitude larger. When comparing the execution time needed for the same error however the fast method needs less computation time for the same error. [21] presents similar results, while also comparing (F)TES4 to the (F)$\mathrm{CF}_2^{[4]}$ method.

All in all, the fast methods yield promising results for the 1D case. Currently, the transmission rates for communication systems using polarization NFDM techniques are limited by the numerical implementation of the NFT for the Manakov equation [12, Sec. 6.4]. The authors of [14] and [11], amongst others, also stated the need for fast NFT algorithms for the Manakov equation in order to make dual polarization NFDM techniques fully competitive with other nonlinear communication techniques. In the remainder of this thesis I will therefore extend these methods to the dual-polarization case, implement them in the open source FNFT library and benchmark them in several testcases.

Chapter 4

Fast NFTs for the Manakov equation

Section 3-3 gave the framework for the Fast Nonlinear Fourier Transform (FNFT) and cited some results for the Nonlinear Schrödinger Equation (NSE). In this chapter we extend these methods to the Manakov equation and develop fast NFT algorithms for Manakov equation. We do this by first addressing the concept of exponential integrators upon which the algorithms rely in Section 4-1. In Sections 4-2 - 4-4 we use coordinate transforms and exponential splittings to bring the exponential integrator methods discussed in Section 3-1-1 in polynomial form. We elaborate on these coordinate transforms in Section 4-5. We address polynomial multiplication and polynomial evaluation in Section 4-6 and Section 4-7 respectively. In Section 4-8 we give a method to determine the discrete eigenvalues. We derive the computational costs for the fast NFT algorithms for the Manakov equation in Section 4-9. Section 4-10 explains how to use Richardson Extrapolation (RE) to further decrease the error of the fast methods, and we close the chapter with a summary of the developed fast NFT algorithm in Section 4-11.

4-1 Exponential integrators and exponential splittings

The methods from the articles cited in Section 3-3-1 and all other fast methods implemented in the FNFT library are fast implementations of exponential integration methods. The basic idea behind an exponential integrator method is to define for a Differential Equation (DE) a related DE which has the same stiffness properties as the original DE and which can be solved exactly [26]. This related DE is often found by linearizing the original DE around a certain state. The exact solution of the related DE will involve one or multiple matrix exponential(s), hence the name exponential methods. Exponential methods are of particular interest because they offer a good trade off between accuracy and computational cost and are fairly easy to implement [10], [6]. Most importantly, if combined with a suitable exponential splitting and coordinate transform, some of them can be brought in polynomial form which in turn makes them good candidates for fast NFT algorithms.
An exponential splitting can be used to approximate the matrix exponential $\mathrm{expm}(\mathbf{C}) =$

$\text{expm}(\mathbf{A}+\mathbf{B})$ such that $\mathbf{A}$ and $\mathbf{B}$ each depend on only one variable. In the context of FNFT specifically, exponential splittings are used to split the transition matrix $\mathbf{\Phi}$ in terms involving either the known samples of $\mathbf{q}(t)$ or the spectral parameter λ: $\text{expm}(\mathbf{\Phi}h) = \text{expm}(\mathbf{A}(\lambda) + \mathbf{B}(\mathbf{q}(t))$. In [28, Eq. 17 - Eq. 24], expressions for exponential splittings are given that result in suitable forms for the FNFT procedure.
In Sections 4-2 - 4-4 we will show for the methods introduced in Section 3-1-1 how to use exponential splittings and a coordinate transform to bring them in polynomial form.

4-2 Boffetta-Osborne method

We introduced the Boffetta-Osborne (BO) method in Section 3-1-1. To bring this method in a form that is a polynomial in the transformed coordinate $z(\lambda)$ we take a look at the transition matrix for this method Eq. (3-8). We split the matrix $\mathbf{P}[n]$ in the sum of 2 matrices where one is dependent on the variable λ and the other depends just on the (known) values of $\mathbf{q}(t)$ for a certain $t = t_n$ as follows:

$$\mathbf{P}[n] = \begin{bmatrix} -j\lambda & q_1[n] & q_2[n] \\ -\kappa q_1^*[n] & j\lambda & 0 \\ -\kappa q_2^*[n] & 0 & j\lambda \end{bmatrix} \tag{4-1}$$

$$= \underbrace{\begin{bmatrix} -j\lambda & 0 & 0 \\ 0 & j\lambda & 0 \\ 0 & 0 & j\lambda \end{bmatrix}}_{\mathbf{A}} + \underbrace{\begin{bmatrix} 0 & q_1[n] & q_2[n] \\ -\kappa q_1^*[n] & 0 & 0 \\ -\kappa q_2^*[n] & 0 & 0 \end{bmatrix}}_{\mathbf{B}[n]} \tag{4-2}$$

As the BO method is a second order method, we want to use a splitting scheme which is also second order accurate or higher. Otherwise the error of the resulting method will be of a lower order. For illustration purposes, we choose the third order splitting [28, Eq. 19].

$$\begin{aligned}
\text{expm}(\mathbf{P}[n]h) =& \text{expm}\left((\mathbf{A}+\mathbf{B}[n])h\right) \\
\approx& \frac{9}{8}\text{expm}\left(\frac{1}{3}\mathbf{A}h\right)\text{expm}\left(\frac{2}{3}\mathbf{B}[n]h\right)\text{expm}\left(\frac{2}{3}\mathbf{A}h\right)\text{expm}\left(\frac{1}{3}\mathbf{B}[n]h\right) - \frac{1}{8}\text{expm}\left(\mathbf{A}h\right)\text{expm}(\mathbf{B}[n]h) \\
=& \frac{9}{8}\begin{bmatrix} \exp(-\frac{1}{3}j\lambda h) & 0 & 0 \\ 0 & \exp(\frac{1}{3}j\lambda h) & 0 \\ 0 & 0 & \exp(\frac{1}{3}j\lambda h) \end{bmatrix}\text{expm}\left(\frac{2}{3}\mathbf{B}[n]h\right) \times \\
& \begin{bmatrix} \exp(-\frac{2}{3}j\lambda h) & 0 & 0 \\ 0 & \exp(\frac{2}{3}j\lambda h) & 0 \\ 0 & 0 & \exp(\frac{2}{3}j\lambda h) \end{bmatrix}\text{expm}\left(\frac{1}{3}\mathbf{B}[n]h\right) \\
& - \frac{1}{8}\begin{bmatrix} \exp(-j\lambda h) & 0 & 0 \\ 0 & \exp(j\lambda h) & 0 \\ 0 & 0 & \exp(j\lambda h) \end{bmatrix}\text{expm}\left(\mathbf{B}[n]h\right).
\end{aligned} \tag{4-3}$$

If we use the coordinate transform $z = \exp(\frac{1}{3}j\lambda h)$ we can write this as a polynomial matrix in z:

$$\begin{aligned}
\text{expm}((\mathbf{A}(\mathbf{z}(\lambda)) + \mathbf{B}[n])h) \approx \frac{9}{8}&\begin{bmatrix} 1/z & 0 & 0 \\ 0 & z & 0 \\ 0 & 0 & z \end{bmatrix} \text{expm}(\frac{2}{3}\mathbf{B}[n]h)\times \\
&\begin{bmatrix} 1/z^2 & 0 & 0 \\ 0 & z^2 & 0 \\ 0 & 0 & z^2 \end{bmatrix} \text{expm}(\frac{1}{3}\mathbf{B}[n]h) \\
&- \frac{1}{8}\begin{bmatrix} 1/z^3 & 0 & 0 \\ 0 & z^3 & 0 \\ 0 & 0 & z^3 \end{bmatrix} \text{expm}(\mathbf{B}[n]h) \\
=\frac{9}{8}&\left(z^{-1}\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\mathbf{E_1}} + z^{1}\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{E_2}} \right) \text{expm}(\frac{2}{3}\mathbf{B}[n]h)\times \\
&\left(z^{-2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + z^{2}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \text{expm}(\frac{1}{3}\mathbf{B}[n]h) \\
&- \frac{1}{8}\left(z^{-3}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + z^{3}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \text{expm}(\mathbf{B}[n]h).
\end{aligned} \tag{4-4}$$

We use $\mathbf{E_1}$ and $\mathbf{E_2}$ to refer to the matrices in the second equation as shown. We can now move multiplication by terms of z in front of the matrices:

$$\begin{aligned}
\text{expm}((\mathbf{A}(\mathbf{z}) + \mathbf{B}[n])h) \approx z^{-3}&\left(\frac{9}{8}\mathbf{E_1}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_1}\text{expm}(\frac{1}{3}\mathbf{B}[n]h) - \frac{1}{8}\mathbf{E_1}\text{expm}(\mathbf{B}[n]h)\right) + \\
&z^{-1}\left(\frac{9}{8}\mathbf{E_2}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_1}\text{expm}(\frac{1}{3}\mathbf{B}[n]h)\right) + \\
&z^{1}\left(\frac{9}{8}\mathbf{E_1}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_2}\text{expm}(\frac{1}{3}\mathbf{B}[n]h)\right) + \\
&z^{3}\left(\frac{9}{8}\mathbf{E_2}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_2}\text{expm}(\frac{1}{3}\mathbf{B}[n]h) - \frac{1}{8}\mathbf{E_2}\text{expm}(\mathbf{B}[n]h)\right).
\end{aligned} \tag{4-5}$$

At this point, multiplication with z^l for a certain $l \in \mathbb{N}$ might be needed to ensure only positive powers of z and yield a polynomial matrix. In this case we multiply with z^3:

$$\begin{aligned} z^3\text{expm}((\mathbf{A}(\mathbf{z}) + \mathbf{B}[n])h) \approx & \left(\frac{9}{8}\mathbf{E_1}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_1}\text{expm}(\frac{1}{3}\mathbf{B}[n]h) - \frac{1}{8}\mathbf{E_1}\text{expm}(\mathbf{B}[n]h)\right) + \\ & z^2\left(\frac{9}{8}\mathbf{E_2}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_1}\text{expm}(\frac{1}{3}\mathbf{B}[n]h)\right) + \\ & z^4\left(\frac{9}{8}\mathbf{E_1}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_2}\text{expm}(\frac{1}{3}\mathbf{B}[n]h)\right) + \\ & z^6\left(\frac{9}{8}\mathbf{E_2}\text{expm}(\frac{2}{3}\mathbf{B}[n]h)\mathbf{E_2}\text{expm}(\frac{1}{3}\mathbf{B}[n]h) - \frac{1}{8}\mathbf{E_2}\text{expm}(\mathbf{B}[n]h)\right). \end{aligned} \tag{4-6}$$

This expression now approximates the transition matrix for a single timestep of BO up to third order accuracy. Only the z is unknown and should be kept as a variable. The other terms can all be computed using the samples of $\mathbf{q}$. The resulting matrix is a polynomial matrix: each element can be written as a polynomial $g = g_0 + g_1 z + g_2 z^2 + \dots g_P z^P$, where P is the degree of the polynomial which is $P = 6$ in this particular case. The MATLAB code used to get the symbolic expressions for the polynomial coefficients g_i in Eq. (4-6) can be found in appendix Section A-1-1. The MATLAB code for getting the polynomial coefficients of methods based on BO with different splittings can be found in Section A-1-2, Section A-1-3, Section A-1-4, and Section A-1-5 respectively.

4-3 $\text{CF}_2^{[4]}$ method

The transition matrix for this method was given in Eq. (3-9). This is a multiplication of two exponential matrices. To put these into polynomial form, we take a look at the first one:

$$\begin{aligned} & \exp(h(a_2\mathbf{A_1} + a_1\mathbf{A_2})) \\ & = \exp\left(a_2\begin{bmatrix} -j\lambda & q_1(t_n^-) & q_2(t_n^-) \\ -\kappa q_1^*(t_n^-) & j\lambda & 0 \\ -\kappa q_2^*(t_n^-) & 0 & j\lambda \end{bmatrix} + a_1\begin{bmatrix} -j\lambda & q_1(t_n^+) & q_2(t_n^+) \\ -q_1^*(t_n^+) & j\lambda & 0 \\ -q_2^*(t_n^+) & 0 & j\lambda \end{bmatrix}\right) \\ & = \exp\left(\begin{bmatrix} -j\lambda/2 & a_2q_1(t_n^-) + a_1q_1(t_n^+) & a_2q_2(t_n^-) + a_1q_2(t_n^+) \\ a_2q_1^*(t_n^-) + a_1q_1^*(t_n^+) & j\lambda/2 & 0 \\ a_2q_2^*(t_n^-) + a_1q_2^*(t_n^+) & 0 & j\lambda/2 \end{bmatrix}\right) \end{aligned} \tag{4-7}$$

where $t_n^- = t_n - \frac{\sqrt{3}}{6}h$ and $t_n^+ = t_n + \frac{\sqrt{3}}{6}h$. The second matrix can be written in an analogous way. We notice that this matrix has the same form as the transition matrix $\mathbf{\Phi}_{BO}$, but with samples $a_2q_k(t_n - \frac{\sqrt{3}}{6}) + a_1q_k(t_n + \frac{\sqrt{3}}{6})$ instead of $q_k(t_n)$ and $\lambda/2$ instead of λ. That means we can implement this method as if it were the BO method, but replace the samples and values of λ with their appropriate expressions. The $\text{CF}_2^{[4]}$ method with D samples then effectively turns into the BO method with 2D samples and transition matrices. Bringing this method in polynomial form is then done in the same way as explained in Section 4-2. However, we cannot use the third-order splitting we used in Section 4-2. We should choose a splitting scheme with an order at least as high as the base method, as otherwise the error introduced by the splitting becomes dominant. For $\text{CF}_2^{[4]}$ we should thus choose at least a fourth-order splitting. The

fourth-order splitting given in [28, Eq. 20] is a suitable candidate. The coordinate transform should then be chosen as $z = \exp(jh\lambda/4)$; this allows us to both represent $\exp(\frac{1}{4}\mathbf{A}h)$ and $\exp(\frac{1}{2}\mathbf{A}h) = \exp(\frac{1}{4}\mathbf{A}h)^2$ in polynomial form. For getting the polynomial coefficients, the same MATLAB code is used as for the methods based on BO and can be found in Section A-1-3 - Section A-1-5.

4-4 TES4 method

From Eq. (3-10) we see that only the second of the three matrix exponentials involves λ. This matrix can thus be split using the same approach as in Section 4-2 and choosing any of the splitting from [28]. However, we would like to illustrate the use of Suzuki splitting [32] here, as that is the splitting used for the fast implementation of TES4 in [22] and [21]. We use the expression for Suzuki factorization as given in [22, Eq. 9]:

$$\begin{aligned}\exp\left(h(\mathbf{A}+\mathbf{B})\right) \approx \exp\left(\frac{7}{48}h\mathbf{B}\right)\exp\left(\frac{1}{3}h\mathbf{A}\right)\exp\left(\frac{3}{8}h\mathbf{B}\right)\exp\left(-\frac{1}{3}h\mathbf{A}\right)\exp\left(-\frac{1}{48}h\mathbf{B}\right)\times \\ \exp\left(h\mathbf{A}\right)\exp\left(-\frac{1}{48}h\mathbf{B}\right)\exp\left(-\frac{1}{3}h\mathbf{A}\right)\exp\left(\frac{3}{8}h\mathbf{B}\right)\exp\left(\frac{1}{3}h\mathbf{A}\right)\exp\left(\frac{7}{48}h\mathbf{B}\right)\end{aligned} \tag{4-8}$$

With coordinate transformation $z = \exp(j\lambda h/3)$ all matrix exponentials involving λ can be represented as polynomials in z. The lowest power of z resulting from multiplying all these matrices is z^{-7}, therefor we should multiply by z^7 to arrive at a polynomial form. MATLAB code for getting the polynomial coefficients is given in Section A-1-6.

4-5 A note on coordinate transforms

All coordinate transforms $z = f(\lambda)$ used for the FNFT should possess the following properties:

- $z(\lambda)$ maps λ onto the unit circle for all $\lambda \in \mathbb{R}$.
- The mapping turns the transition matrix $\mathbf{\Phi}[n, \lambda]$ into a 3×3 matrix $\mathbf{G}[n, z]$, where each entry is a rational function in z. Multiplication with z^l for a certain $l \in \mathbb{N}$ might be needed to ensure only positive powers of z needed for a polynomial.

Indeed, all coordinate transformations in sections Section 4-2, Section 4-3 and Section 4-4 are of the form $z = \exp(j\lambda h/m)$ and thus map λ to a z on the unit circle in the complex plane. A suitable coordinate transform for the methods with sixth-order splitting is $z = \exp(j\lambda h/6)$. For all other splittings proposed in [28, Eq. 17 - Eq. 24] it is also possible to choose a transformation of that form such that the resulting matrices are polynomial in z.
Mapping $z(\lambda)$ to the unit circle offers multiple benefits: The norm of z^p will always be 1, independent of p. So even for high powers of z that will arise when multiplying a high number transition matrices, the absolute value of z^p stays bounded. This prevents overflow errors that might occur otherwise in the numerical implementation of this algorithm. Furthermore, the chirp-Z transform can be used to evaluate the polynomials (see also Section 4-7). The polynomial form also allows for fast polynomial multiplication of the transition matrices which will be discussed in the next section.

4-6 Polynomial multiplication

Now that we know how to write the aforementioned methods as polynomial methods we will illustrate how to use the FFT to efficiently multiply the matrices. We will illustrate how this works for the Manakov equation. Let $\mathbf{G}[n, z]$ be the polynomial approximation of the transition matrix after applying the coordinate transform, the splitting scheme and multiplication with z^l. Eq. (3-16) then becomes

$$\mathbf{v}[D, \lambda] = \left(\Pi_{n=1}^{D} z^{-l}\mathbf{G}[n, z]\right)\mathbf{v}[0, \lambda] = z^{-lD}\mathbf{G}(z)\mathbf{v}[0, \lambda] \tag{4-9}$$

The extra term z^{-l} is introduced because we multiplied the transition matrix for a single time step by z^l for a certain $l \in \mathbb{N}$. For the BO method with third-order splitting for example we multiplied with z^3 (see Section 4-2).

We get numerical approximations of the Nonlinear Fourier Transform (NFT) coefficients from Eq. (3-4)

$$\hat{a}(\lambda) = \left[z^{-lD}\mathbf{G}(z(\lambda))\mathbf{v}[0, \lambda]\exp\left(j\lambda(T_2 + \frac{1}{2}h)\right)\right]_1, \tag{4-10}$$

$$\hat{\mathbf{b}}_{\mathbf{k}}(\lambda) = \left[z^{-lD}\mathbf{G}(z(\lambda))\mathbf{v}[0, \lambda]\exp\left(-j\lambda(T_2 + \frac{1}{2}h)\right)\right]_{k+1}, \tag{4-11}$$

where the subscripts 1, $k+1$ denote the first and $k+1$th entry of the resulting vectors. The multiplication of the D matrices in Eq. (4-9) is done in a tree-wise manner, multiplying 2 terms at a time and doing the same for the resulting terms until all terms are multiplied. This is schematically represented in Figure 4-1.

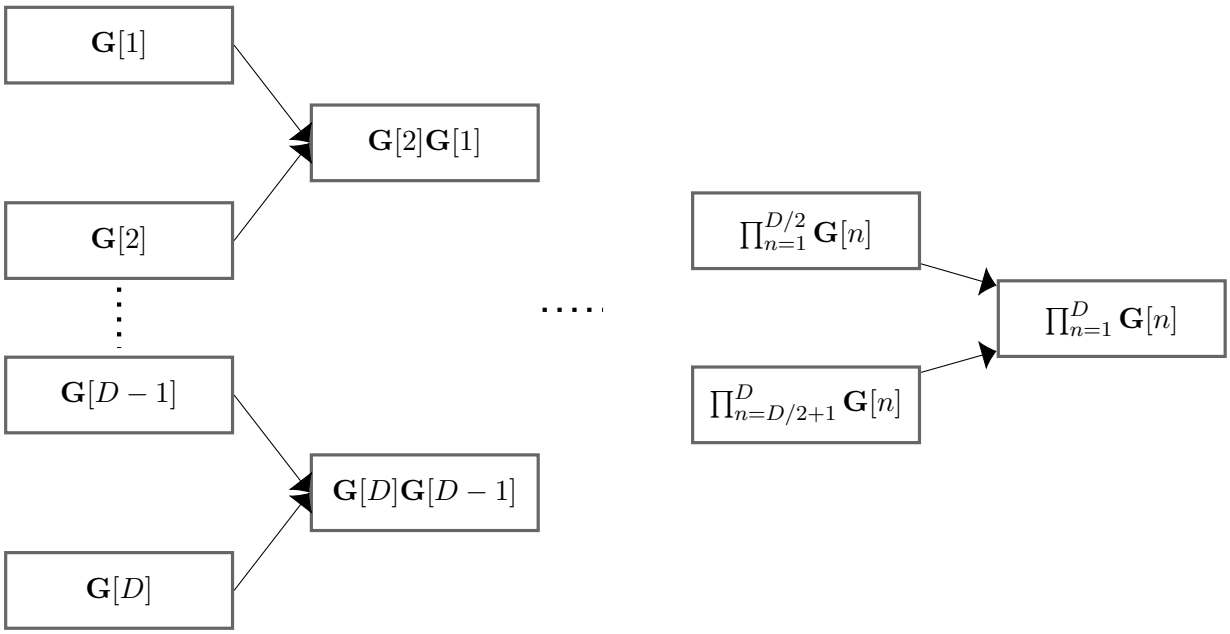


Figure 4-1: Schematic representation of the tree-wise multiplication of polynomial matrices $\mathbf{G}[n]$

Here it is important to keep in mind the order of multiplication and always put the transition matrices for the higher time indices on the left. If the number of samples D is a power of 2,

tree-wise multiplication can be used directly. If not, we can pad the number of matrices with identity matrices such that the total number of matrices is a power of D and apply tree-wise multiplication as before. If we take a look at for example the first 2 transition matrices, we see that each multiplication of a pair of matrices involves 27 polynomial multiplications:

$$\underbrace{\begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix}}_{\mathbf{G}[2]} \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\mathbf{G}[1]} \tag{4-12}$$

$$= \begin{bmatrix} g_{11}h_{11} + g_{12}h_{21} + g_{13}h_{31} & g_{11}h_{12} + g_{12}h_{22} + g_{13}h_{32} & g_{11}h_{13} + g_{12}h_{23} + g_{13}h_{33} \\ g_{21}h_{11} + g_{22}h_{21} + g_{23}h_{31} & g_{21}h_{12} + g_{22}h_{22} + g_{23}h_{32} & g_{21}h_{13} + g_{22}h_{23} + g_{23}h_{33} \\ g_{31}h_{11} + g_{32}h_{21} + g_{33}h_{31} & g_{31}h_{12} + g_{32}h_{22} + g_{33}h_{32} & g_{31}h_{13} + g_{32}h_{23} + g_{33}h_{33} \end{bmatrix}$$

These multiplications are carried out efficiently using the Fast Fourier Transform (FFT) [13]. This approach is used in articles [38], [10], and it is also the multiplication method implemented in the FNFT library. With the help of the FFT two polynomials of at most degree p can be multiplied in $O\left(p\log(p)\right)$ operations [10, Sec. IV].

We calculate the product of g_{11} and h_{11} as an example (we will write g and h here instead of g_{11} and h_{11} to reduce the number of subscripts). We can represent the polynomial $g(z) = g_0 + g_1 z + \dots g_P z^P$ in different ways:

- **Coefficient vector**
  We simply take all coefficients of g_i of the polynomial $g = [g_0 \quad g_1 \quad \dots \quad g_P]^T$, P being the degree of the polynomial.

- **Sample representation**
  The following sample points of the polynomial are given: $(z_0, y_0), \quad (z_1, y_1), \quad \dots \quad (z_p, y_p)$ with $g(z_i) = y_i \quad \text{for} \quad i = 0, 1, \dots, p$. If we know that the function through these points is a polynomial and $z_i \neq z_j$ for $i \neq j$, these samples uniquely determine a polynomial of degree p by the fundamental theorem of algebra.

The expressions for $h(z)$ are analogous. The coefficient representation is the most convenient representation for polynomial evaluation later in the algorithm. It is also the representation in which the elements of the transition matrix are given for all methods implemented in the library. However, polynomial multiplication is more efficient in sample representation: $O(p)$ versus $O(p^2)$ operations for two polynomials of at most degree p. The main idea behind FFT based polynomial multiplication is to efficiently convert the polynomial in coefficient representation to sample representation, carry out the multiplication and switch back to coefficient representation. The conversion between the two representations is done by the Discrete Fourier Transform, which can be efficiently computed (in $O\left(p\log(p)\right)$ FLOPs [13]) using the FFT. For more details we refer to [13]. A schematic representation of the multiplication of two polynomials $g(x)$ and $h(x)$ of at most degree p along with computational costs is given in Figure 4-2.

In this particular case of multiplying 3×3 matrices, we notice that adding two polynomials can be done both in their coefficient as well as in their sample representation. Therefore we

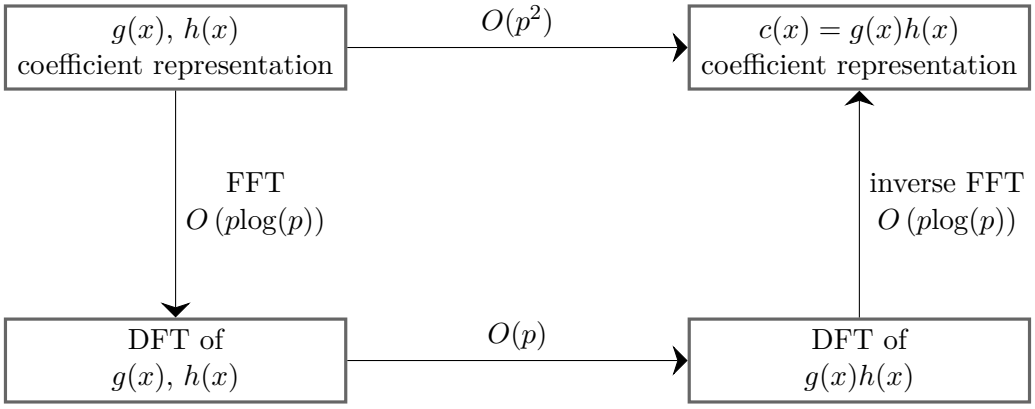


Figure 4-2: Schematic representation of polynomial multiplication using the FFT

only transform back to the coefficient representation after we added all terms for an element of the matrix, saving us two inverse FFTs for each matrix element. The process for the upper left element in our case is schematically shown in Figure 4-3

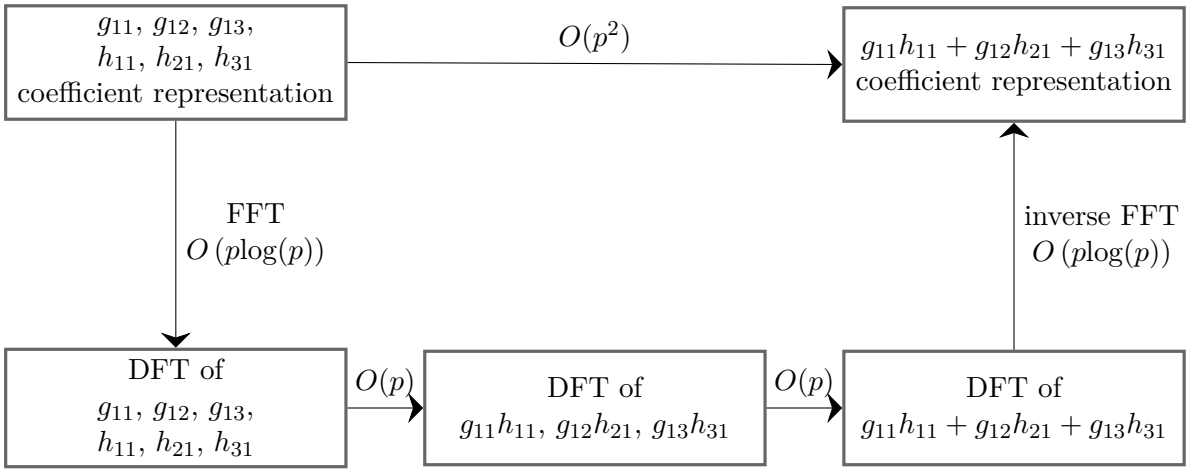


Figure 4-3: Schematic representation of polynomial multiplication using the FFT for one element of a 3×3 matrix

4-7 Polynomial evaluation (chirp-Z transform)

The last step in getting the values of NFT coefficients $a(\lambda)$, $b(\lambda)_{1,2}$ is the evaluation of Eq. (4-10) and Eq. (4-11). Because we take $\mathbf{\Psi^N} = [1 \quad 0 \quad 0]^T \exp(-j\lambda t)$ from Eq. (2-19) as the initial condition, only the first column of $\mathbf{G}(z(\lambda))$ needs to be evaluated as the other elements cancel when multiplied with $\mathbf{\Psi^N}$. All elements of the matrix $\mathbf{G}(z(\lambda))$ are polynomials and we can use the chirp-Z transform algorithm [29] to evaluate them at the desired $z(\lambda)$. This way all values of the NFT coefficients at the nodes specified below can be computed in $O\left((P+M)\log(P+M)\right)$ operations, as opposed to $O(PM)$ operations for the naive approach, where P is the degree of the resulting polynomial and M the number of $z(\lambda)$ for which we

evaluate the polynomial.
In general, the chirp-Z transform efficiently computes the Z-transform $X(z)$ of a sequence of numbers $x(n)$ on M points on the real line:

$$X_k = \sum_{n=0}^{P} x(n)(AW^{-k})^n \quad \text{with } k = 0, 1, \dots M-1. \tag{4-13}$$

Here A can be understood as the starting point of the spiral, and W is the ratio between the points in the complex plane.

In this specific case we have polynomial entries of the total transition matrix $g = g_0 + g_1 z + g_2 z^2 + \dots g_P z^P$ that we want to evaluate on M points $z = f(\lambda)$ corresponding to M equidistant points $\lambda \in [\Xi_0, \Xi_1]$ in the complex plane:

$$\lambda_0 = \Xi_0, \lambda_1 = \Xi_0 + d\lambda, \lambda_2 = \Xi_0 + 2d\lambda, \dots \lambda_{M-1} = \Xi_0 + (M-1)d\lambda, \tag{4-14}$$

where

$$\Delta\lambda = \frac{\Xi_1 - \Xi_0}{M-1}. \tag{4-15}$$

From the first property of the coordinate transforms mentioned in Section 4-5 we know that λ is mapped to the unit circle: all points we wish to evaluate lie on a spiral in the Z-plane with constant radius 1.
Let $x(n)$ in Eq. (4-13) be the polynomial coefficient for z^n, $x(n) = g_n$, and $z_k = \exp(j\lambda_k h/m)$. We can then choose $A = \exp(j\Xi_0 h/m)$ and $W = \exp(-jd\lambda h/m)$. Then we have

$$\begin{aligned} AW^{-k} &= \exp(j\Xi_0 h/m)\left(\exp(-jd\lambda h/m)\right)^{-k}, \\ &= \exp(j\Xi_0 h/m)\exp(jd\lambda k h/m), \\ &= \exp(j(\Xi_0 + d\lambda k)h/m), \\ &= \exp(j\lambda_k h/m) = z_k, \end{aligned} \tag{4-16}$$

which means that by applying the chirp-Z transform we are exactly evaluating the polynomial g for all M points in the complex plane corresponding to the desired λ's. The key point here is that this happens for all M points at once with one chirp-Z transform.

Once the first column of $\mathbf{G}(z(\lambda))$ is evaluated we should multiply the values by $\exp(-j\lambda t)$ for the initial condition, by $z^{-lD} = \exp(-j\lambda hlD/m)$ to compensate for the multiplication with z^l and by $\exp\left(j\lambda(T_2 + \frac{1}{2}h)\right)$ or $\exp\left(-j\lambda(T_2 + \frac{1}{2}h)\right)$ for the a and $b_{1,2}$ coefficients respectively. These multiplications do need to be carried out for each λ separately, but they can be combined in one step and just add a phase factor.

4-8 Determining the eigenvalues of NFT coefficient a

As mentioned in Section 3-1, numerically finding the discrete spectrum of the Manakov equation is beyond the scope of this thesis project. We will however briefly address how to find the roots of NFT coefficient $a(\lambda)$. These discrete eigenvalues offer interesting insights in itself as they are the λ-values for which solitons (see Section 2-2-1) occur. Computing the discrete eigenvalues would also be the first step if one were to compute the discrete spectrum.

From Section 4-7 we can write down the expression for $a(\lambda)$:

$$\begin{aligned} a(\lambda) &= z^{-lD}\mathbf{G}(z(\lambda))_{11}\mathbf{v}_1[0]\exp\left(j\lambda(T_2+\frac{1}{2}h)\right) \\ &= \exp(-j\lambda hlD/m)\mathbf{G}(z(\lambda))_{11}\exp\left(-j\lambda(T_1-\frac{1}{2}h)\right)\exp\left(j\lambda(T_2+\frac{1}{2}h)\right) \end{aligned} \tag{4-17}$$

All exponential terms just add a phase factor and do not influence the location of the roots of the polynomial $a(\lambda)$: we can therefore look at the roots of the function $\mathbf{G}(z(\lambda))_{11}$ in terms of z and then convert those to λ.

This also shows the advantage of fast methods when finding the discrete spectrum: Unlike for the slow methods, where we choose a λ and get the a for this specific λ, we get a polynomial approximation of $a(\lambda)$ where λ is still a variable. This means that we can use a fast polynomial rootfinder method to find approximations of the roots of a. Details on the fast polynomial rootfinder used in the FNFT library can be found in [5].

The polynomials we are working with merely approximate the roots of $a(\lambda)$, and it is possible to find roots of this polynomial that are not discrete eigenvalues. The roots therefore need to be filtered. The most basic filtering is to discard all values in the lower half complex plane. We can do this because of the symmetry mentioned in Section 2-2: if λ is an eigenvalue, then so in λ^*. After this, the remaining roots are merged: if two roots are less than the machine precision apart, it is assumed they are the same root.

4-9 Computational complexity of the fast methods

In this section we derive the computational costs of the the fast NFT algorithms. The pseudocode for the fast methods is given in Algorithm 2.

Algorithm 2 Computing the continuous nonlinear spectrum using using a fast numerical NFT method

Input: $\mathbf{q}[1],...,\mathbf{q}[D]$, T_1, T_2, desired λ-interval, M
Output: $a[1],...a[M]$, $\mathbf{b}[1],...\mathbf{b}[M]$
for n=1:D **do**
  Determine $\mathbf{G}[n,z(\lambda)]$ based on the transition matrix $\mathbf{\Phi}(t[n],\lambda)$ (note here that $\mathbf{G}$ is still a function of λ: we do not need to specify λ)
end for
Fast multiplication: $\mathbf{G}(z) \leftarrow \Pi_{i=1}^{D}\mathbf{G}[n,z]$
Fast evaluation of $\mathbf{G_{11}}(z)$, $\mathbf{G_{21}}(z)$ and $\mathbf{G_{31}}(z)$ for all M values of $z(\lambda)$
for i=1:M **do**
  $a[i] = z[i]^{-lD}\mathbf{G_{11}}(z[i])\exp\left(j\lambda(T_1-\frac{1}{2}h)\right)\exp\left(j\lambda(T_2+\frac{1}{2}h)\right)$
  $b_1[i] = z[i]^{-lD}\mathbf{G_{21}}(z[i])\exp\left(j\lambda(T_1-\frac{1}{2}h)\right)\exp\left(-j\lambda(T_2+\frac{1}{2}h)\right)$
  $b_2[i] = z[i]^{-lD}\mathbf{G_{31}}(z[i])\exp\left(j\lambda(T_1-\frac{1}{2}h)\right)\exp\left(-j\lambda(T_2+\frac{1}{2}h)\right)$
end for

One notes here that, instead of doing the whole iteration for each $\lambda[j]$, we only need to evaluate the polynomial matrix $\Phi(\lambda[j])$ and do one multiplication with a phase factor for each λ.

The pseudocode and computational cost for the matrix multiplication step are given in Algorithm 3. The costs for evaluating the polynomials at M points were given in Section 4-7. The maximum degree of the resulting polynomial is $P = pD$, where p is the maximum degree of the individual polynomials at the start and D the number of polynomials. The cost of evaluating the polynomials is thus $O\left((pD + M)\log(pD + M)\right)$, which makes the total cost of the FNFT $O\left(\underbrace{D\log^2(D)}_{\text{multiplication cost}} + \underbrace{\left((pD + M)\log(pD + M)\right)}_{\text{evaluation cost}}\right) = O\left(D\log^2(D)\right)$.

Algorithm 3 Multiplying the transition matrices using tree-wise multiplication

Input:$\mathbf{G}[1, z], ..., \mathbf{G}[D, z]$ transition matrices
Output: total transition matrix $\Pi_{i=1}^{D}\mathbf{G}[n, z]$
Complement the number of matrices D with identity matrices until we arrive at D_c matrices with $D_c = 2^r$ for some $r \in \mathbb{N}$
while $D_c > 1$ **do**
  for $n = 1 : D_c/2$ **do**
    $\mathbf{G}[n, z] \leftarrow \mathbf{G}[2n, z]\mathbf{G}[2n - 1, z]$
  end for
  $D_c \leftarrow D_c/2$
end while

To determine the computational costs of the multiplication step we first look at the inner loop of the pseudocode where we do 27 polynomial multiplications using the FFT. Let p be the maximum degree of the polynomials and D_c the number of matrices. From Section 4-6 we know that the cost for multiplying two polynomials of degree p is $O\left(p\log(p)\right)$. We now look at the outer loop. For the first iteration of the outer loop we carry out the inner loop $D_c/2$ times, so the computational cost is $O\left(\frac{D_c}{2}p\log(p)\right)$. For the second iteration of the outer loop, we have only half the number of matrices, but the maximum degree of the polynomials has doubled, which leads to a computational cost of $O\left(\frac{D_c}{4}2p\log(2p)\right) = O\left(\frac{D_c}{2}p\log(2p)\right)$. For the iteration after that the cost will be $O\left(\frac{D_c}{2}p\log(4p)\right)$ and so on. We carry out the outer loop $\log_2(D_c)$ times, so on the last iteration the cost is $O\left(\frac{D_c}{2}p\log(2^{\log_2(D_c)}p)\right) = O\left(\frac{D_c}{2}p\log(D_cp)\right)$: this is the maximum cost of one iteration of the outer loop, and to get an upper bound on the computational cost we will assume that all iterations of the outer loop have this cost. The total cost of the multiplication is thus $\log_2(D_c)O\left(\frac{D_c}{2}p\log(D_cp)\right) = O\left(\frac{D_c}{2}p\log(D_cp)\log(D_c)\right)$. As p is constant once we have chosen the method we can ignore it when considering the computational costs. Therefore we say that the computational cost of the matrix multiplication using the FFT is $O\left(D\log^2(D)\right)$.

4-10 Richardson Extrapolation

One technique we can use to make the algorithm more accurate is RE. With RE we carry out the FNFT procedure a second time using only half of the original samples and then use knowledge of the order of the numerical method to increase the order of that method by one.

We refer to [37, Sec. 3.7] for the derivation but give the expression here:

$$\hat{\rho}_{\text{new}} = \frac{2^r \hat{\rho}_{\text{old}}(h) - \hat{\rho}_{\text{old}}(2h)}{2^r - 1}. \tag{4-18}$$

$\hat{\rho}(2h)$, $\hat{\rho}(h)$ are the numerical approximations of ρ for time step size $2h$ and h respectively. Choosing a time step size of $2h$ corresponds to using only half the samples. r is the order of the chosen numerical method. Instead of the reflection coefficient ρ this procedure can also be applied to any of the NFT coefficients a of $b_{1,2}$.
An advantage of using this procedure is that an additional accuracy is achieved without choosing a higher order method or using more samples. A disadvantage is that this of course adds more computation time. Another disadvantage is that the order of the numerical method needs to be known: even if the order is known in theory, the actual error decay will only be of order r within a certain region. Outside that region discretization errors will be dominant for larger h and rounding errors will be dominant for smaller h. If the actual order does not match the assumed order r, the result from Richardson extrapolation will be less accurate than the original result for time step h.

4-11 Summary of the FNFT algorithm

We can summarize the FNFT algorithm in the following steps:

- Choose a suitable exponential integration method
- Use a coordinate transform and an exponential splitting to turn the transition matrix in a polynomial matrix and map all real λ in the frequency domain onto the unit circle
- Multiply all polynomial matrices using the FFT
- Evaluate the resulting polynomial for the desired points in the frequency domain using the chirp-Z transform
- If desired, find the discrete eigenvalues using a fast rootfinder algorithm on the polynomial $\mathbf{G}(z(\lambda))_{11}$

Chapter 5

Overview of developed algorithms

In Section 3-3 we mentioned the FNFT method for the Nonlinear Schrödinger Equation (NSE), which was developed to solve the problem of numerical NFT algorithms being too slow. In Chapter 4 we extended these ideas from the NSE with one space coordinate to the case with two space coordinates and developed fast NFT algorithms for the Manakov equation that compute

- the continuous spectrum using the BO discretization and various exponential splitting schemes
- the continuous spectrum using the $\mathrm{CF}_2^{[4]}$ discretization and various exponential splitting schemes
- the continuous spectrum using the TES4 discretization and Suzuki factorization
- the discrete eigenvalues using the above mentioned discretizations and splittings and a fast polynomial rootfinder

To make the developed algorithms widely available, we implemented them in the open-source Fast Nonlinear Fourier Transform (FNFT) software library that Wahls et. al. have been working on [39]. Currently, fast methods have been implemented to compute the continuous and discrete spectrum for the NSE with vanishing and (quasi) periodic boundary conditions and to compute the reflection coefficient ρ of the Korteweg-de Vries (KdV) equation with vanishing boundary conditions. See [2] for the GitHub repository of this library. On the development branch of this repository, code for computing the discrete spectrum of the KdV can also found.
The second goal of this thesis is to implement the algorithms developed in Chapter 4 and incorporate them in the FNFT library. The existing code for the NSE with vanishing boundary conditions served as the basis for the part of the code concerning the Manakov equation. This chapter explains how I implemented the algorithms and highlights my contributions to the library.

5-1 Structure of the code contributed to the FNFT library

Figure 5-1 shows a diagram with the most important functions in the library concerning the Manakov equation. The source code can be found on GitHub: https://github.com/ldvries/FNFT/tree/development. At the time of writing, the new features for the Manakov Equation (ME) had not been merged with the existing library yet, but they will be in the future. They can then be found on https://github.com/FastNFT/FNFT/tree/development. The prefixes fnft___ and fnft_ that are found in the source code have been omitted in the diagram. fnft_ (with one underscore) is used for the highest level functions; in the diagram, only "manakovv" has this prefix. fnft___ (with two underscores) denote functions accessed by the higher level functions, but that will most likely not be accessed directly by a user of the library. The extra "v" in "manakovv" indicates that this code is for vanishing boundary conditions (that is, $\mathbf{q}(x,t)$ satisfies Assumption 1 Section 2-2). The code without the extra "v" should work as well for cases with periodic boundary conditions if they were to be implemented in the future. Computing the NFT of the Manakov equation using the FNFT library is schematically shown in Figure 5-1. Arrowheads pointing from function A to function B mean that function A calls function B.

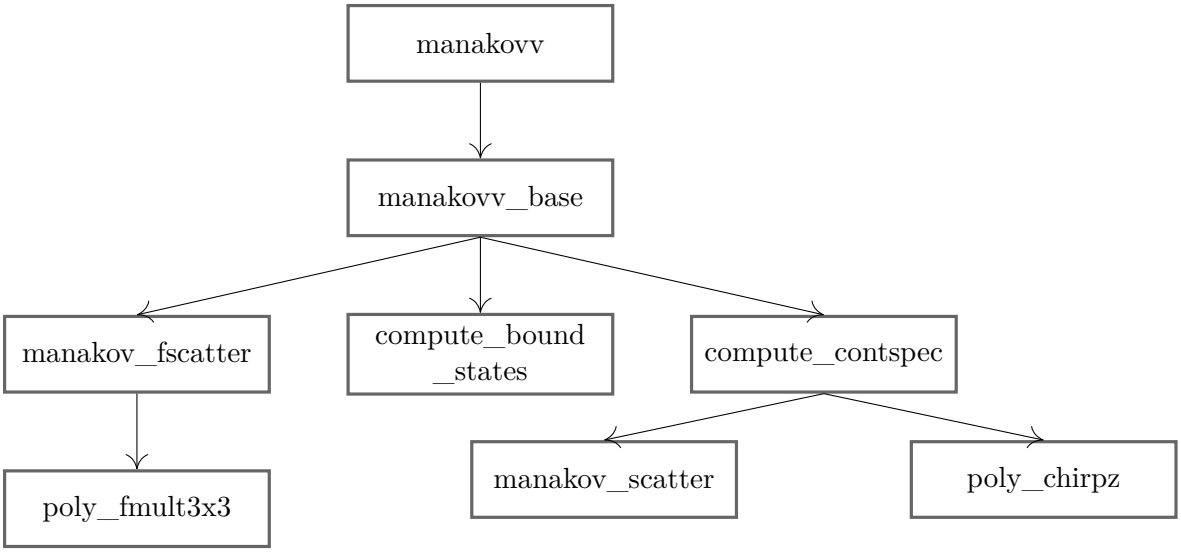


Figure 5-1: Diagram of the most important functions in the library for the Manakov FNFT

- The user calls "manakovv" and provides all necessary parameters such as the number of samples, sample points, values of the samples and desired points in the frequency domain λ for which to calculate the continuous spectrum.
- "manakovv" checks these inputs, performs some preprocessing of the signal (determining the interpolated samples for the methods based on $\mathrm{CF}_2^{[4]}$) and prepares to call manakovv_base
- manakovv_base computes the polynomial transition matrix by calling manakov_fscatter if a fast method is chosen. If a slow method is chosen, it leaves the transition matrix variable empty.

- manakov_fscatter computes the transition matrix as outlined in Section 3-3. It first computes the transition matrix for each step (see Section 4-2, Section 4-3 and Section 4-4) and then calls manakovv_poly_fmult3x3 to perform multiplication using the FFT and treewise multiplication (see Section 4-6).
- This transition matrix is then passed from manakovv_base to compute_contspec. If compute_contspec recieves a transition matrix (i.e. the chosen discretization is a fast method) it calls poly_chirpz, which evaluates this transition matrix for different λ as explained in Section 4-7. compute_contspec then computes the continuous spectrum (reflection coefficient ρ, a and $b_{1,2}$ coefficient, or both) as outlined in Section 4-7 and returns the continuous spectrum. If it receives an empty variable for the transition matrix, i.e. a slow method was chosen, it calls manakovv_scatter for each desired λ. manakovv_scatter then performs the one-step iteration Eq. (3-3) M times to arrive at the continuous spectrum.
- If the user wants to compute the discrete eigenvalues, the total transition matrix is also passed to compute_bound_states. The bound states (discrete eigenvalues) are then computed as explained in Section 4-8.

A more detailed diagram and description of all relevant functions can be found in the appendix Section B-0-1.

5-1-1 Naming conventions in the library

The fast methods in the library based on BO and $\text{CF}_2^{[4]}$ are all named 2splitYZ and 4splitYZ. "2" or "4" denote the order of the base method, 2 for BO and 4 for $\text{CF}_2^{[4]}$. "Y" indicates the order of the splitting scheme chosen from [28], and "Z" can be either "A" or "B": "A" for the splitting scheme as it appears in [28], "B" for the splitting scheme with the roles of matrices $\mathbf{A}$ and $\mathbf{B}$ swapped. Here $\mathbf{C}$ is the matrix to be split and $\mathbf{C} = \mathbf{A} + \mathbf{B}$ as in Section 4-1. The discretization described in Section 4-2 is thus the 2split3A method.

5-2 Methods implemented in the library

The library currently offers a wide range of discretizations to determine the Nonlinear Fourier Transform (NFT) of the NSE and the KdV equation. I chose not implement all of those, but rather those that are now known to be the most efficient for the NSE or might offer some interesting insights during the testing phase. I wanted to cover a couple of points:

- Investigate if using a fast NFT algorithm for the Manakov equation improves the error / runtime trade-off compared to using a slow method, and if so, in which cases.
- Investigate the effect of the splitting method on the performance.
- Implement the methods which have the best trade-off between speed and accuracy for the NSE. We expect that these are also the most efficient methods for the Manakov equation and they will be the most useful methods for users of the software.

- Implement a method not based on BO or $CF_2^{[4]}$ by a different research group. Comparisons between algorithms pioneered by the same authors or research groups are quite common, as authors naturally build on their previous work and include a comparison to their own older algorithms. Comparisons to other algorithms developed by other authors are not always included, and including them here will give the reader another point of reference and hopefully allow users of the methods to make a more informed choice.

5-2-1 Motivation of implemented methods

To show the efficiency of the FNFT algorithm, I implemented both the BO method and the $CF_2^{[4]}$ method with their fast counterparts, the 2splitYZ and 4splitYZ methods respectively. To cover the second point, I implemented different order splitting methods for both 2splitYZ (2nd, 4th and 6th order) and 4splitYZ (4th and 6th order). I also implemented both the "A" variant, where the matrices $\mathbf{A}$ and $\mathbf{B}$ are as they appear in [28, Eq. 17 - 24], and the "B" variant, where the matrices are swapped.
Which method is the best choice will of course depend on the specific application. If we just have a small number of samples from the signal, and are only interested in the NFT for a small number of λ's, a slow method might even be the best choice. However, in general the best methods are those of a higher order (order 4, to increase the accuracy) and with a lower number of polynomial coefficients per transition matrix (this keeps the computation time lower). The order of the method depends on the base method provided a splitting of at least the same order is chosen. The number of polynomial coefficients is dependent on the splitting itself. Therefore I chose to implement 4split4B and 4split6B. These are based on $CF_2^{[4]}$ and are thus fourth-order methods. The B variants in this case lead to transition matrices of a lower degree for each step than the A variants.
To satisfy the last point, I chose to implement the FTES4 method [22], [24]. The authors used Suzuki factorization as their exponential splitting method, so I also implemented FTES4 with this splitting method.

Chapter 6

Tests and comparisons

In this chapter we test and benchmark the implemented fast NFT algorithms. To this end we use potential functions $\mathbf{q}(x,t)$ for which the Nonlinear Fourier Transform (NFT) can be determined analytically. For two of the test cases, the single and double soliton, the solutions were already known from literature. They are given in Section 6-1-2 and Section 6-1-3. We also determine the exact solution of two test cases: rectangle potential in Section 6-1-1 and the generic secant hyperbolic potential in Section 6-1-2. These analytic solutions are new to the best of our knowledge. We also give the parameters we used for the test cases in those sections. in Section 6-1-5 we derive restrictions on the step size h and parameters in the spectral domain for numerical NFT of the Manakov equation. In Section 6-2 we show and discuss the results of the tests.

6-1 Test functions

6-1-1 Rectangle

This potential function is given by

$$\mathbf{q}(t) = \begin{cases} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} & \text{for } L_1 \leq t \leq L_2, \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \text{elsewhere.} \end{cases} \tag{6-1}$$

As far as we are aware, no analytic derivation of the NFT coefficients for this potential function is available in the literature. We therefore derive the solution here.

For potential Eq. (6-1), Eq. (2-10) becomes a piecewise constant matrix and Eq. (2-6) becomes

$$\begin{aligned}\mathbf{v_t} &= \begin{cases} \mathbf{P_L v} & \text{for } L_1 \leq t \leq L_2, \\ \mathbf{P_0 v} & \text{elsewhere} \end{cases} \\ \mathbf{P_L} &= \begin{bmatrix} -j\lambda & A_1 & A_2 \\ -\kappa A_1^* & j\lambda & 0 \\ -\kappa A_2^* & 0 & j\lambda \end{bmatrix}, \\ \mathbf{P_0} &= \begin{bmatrix} -j\lambda & 0 & 0 \\ 0 & j\lambda & 0 \\ 0 & 0 & j\lambda \end{bmatrix}. \end{aligned} \tag{6-2}$$

Because $\mathbf{P}$ is piecewise constant, the exact solution in these regions is known to be a matrix exponential. In the region $-\infty \leq t \leq L_1$ we use the boundary condition from Eq. (2-14):

$$\begin{aligned} \mathbf{v}(t) = \text{expm}(\mathbf{P_0}t)\mathbf{c_1} = \begin{bmatrix} \exp(-j\lambda t) & 0 & 0 \\ 0 & \exp(j\lambda t) & 0 \\ 0 & 0 & \exp(j\lambda t) \end{bmatrix} \mathbf{c_1}, \quad \lim_{t\to-\infty} \mathbf{v}(t) = \begin{bmatrix} \exp(-j\lambda t) \\ 0 \\ 0 \end{bmatrix} \\ \Rightarrow \mathbf{c_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{v}(t) = \begin{bmatrix} \exp(-j\lambda t) \\ 0 \\ 0 \end{bmatrix} \text{ for } -\infty \leq t \leq L_1 \end{aligned} \tag{6-3}$$

From this equation we get the boundary condition at $t = L_1$ for the region $L_1 \leq t \leq L_2$:

$$\begin{aligned} \mathbf{v}(t) = \text{expm}(\mathbf{P_L}t)\mathbf{c_2}, \quad \mathbf{v}(L_1) = \begin{bmatrix} \exp(-j\lambda L_1) \\ 0 \\ 0 \end{bmatrix} \\ \Rightarrow \mathbf{c_2} = \text{expm}(\mathbf{P_L}L_1)^{-1}\mathbf{v}(L_1), \quad \mathbf{v}(t) = \text{expm}(\mathbf{P_L}t)\text{expm}(\mathbf{P_L}L_1)^{-1}\mathbf{v}(L_1) \text{ for } L_1 \leq t \leq L_2 \end{aligned} \tag{6-4}$$

This will in turn give us $\mathbf{v}(L_2)$, which serves as the boundary condition for the last region $L_2 \leq t \leq \infty$. We arrive at

$$\begin{aligned} \mathbf{v}(t) = \text{expm}(\mathbf{P_0}t)\mathbf{c_3}, \quad \mathbf{v}(L_2) = \text{expm}(\mathbf{P_L}L_2)\text{expm}(\mathbf{P_L}L_1)^{-1}\mathbf{v}(L_1) \\ \Rightarrow \mathbf{c_3} = \text{expm}(\mathbf{P_0}L_2)^{-1}\text{expm}(\mathbf{P_L}L_2)\text{expm}(\mathbf{P_L}L_1)^{-1}\mathbf{v}(L_1). \end{aligned} \tag{6-5}$$

To get the NFT coefficients we use Eq. (2-25) where we use $\mathbf{\Psi^N} = \mathbf{v} = \text{expm}(\mathbf{P_0}t)\mathbf{c_3}$. We notice that the element in the first row of $\text{expm}(\mathbf{P_0}t)$ cancels against $\exp(j\lambda t)$ for the a coefficient and the elements in the second and third row cancel against $\exp(-j\lambda t)$ for the b

coefficients:

$$\begin{aligned}\operatorname{expm}(\mathbf{P_0}t)\mathbf{c_3}\exp(j\lambda t) &= \begin{bmatrix}1 & 0 & 0\\ 0 & \exp(j\lambda t)^2 & 0\\ 0 & 0 & \exp(j\lambda t)^2\end{bmatrix}\mathbf{c_3}\\ \Rightarrow \operatorname{expm}(\mathbf{P_0}t)\mathbf{c_3}\exp(-j\lambda t) &= \begin{bmatrix}\exp(-j\lambda t)^2 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\mathbf{c_3}\\ &\Rightarrow \begin{bmatrix}a\\ b_1\\ b_2\end{bmatrix} = \mathbf{c_3}.\end{aligned} \tag{6-6}$$

This potential function is interesting because it has a jump at L_1 and L_2, which might lead to numerical issues. We implemented the following case:

rectangle test case:

$$\mathbf{q}(t) = \begin{cases}\begin{bmatrix}0.8\\ 5.2\end{bmatrix} & \text{for } -2 \leq t \leq 2,\\ \begin{bmatrix}0\\ 0\end{bmatrix} & \text{elsewhere.}\end{cases}$$

$$T = [-2 - 0.5h, 2 - 0.5h], \quad \lambda \in [-250, 250], \quad \kappa = +1$$

$T = [T_1, T_2]$ indicates the time domain where the samples of $\mathbf{q}(t)$ are taken. This choice for the time domain ensures that the jump in the potential function is accurately captured. Our choice for the the λ domain does not influence the accuracy of the results as T does. We should merely make it large enough such that we capture all interesting dynamics. If we look at the exact solution we observe that $a \to 1$ and $b_{1,2} \to 0$ as $\lambda \to \pm\infty$, and thus $\rho_{1,2} \to 0$ in these limits. However, the a and $b_{1,2}$ coefficients tend to their asymptotic values quite slowly and oscillate. Therefor a relatively large domain for λ with a fine mesh to capture all oscillations makes sense. We somewhat arbitrarily define the region of interesting dynamics to be the region where there are values of $\rho_2 > 2/100$, 2 being the approximate maximum value ρ_2 reaches in the region around 0. That happens for $\lambda \in [-247, 247]$, so we round that up to $\lambda \in [-250, 250]$. The fine grid size is taken care of automatically for larger D as we choose $M = D$.

6-1-2 Single soliton

For the single soliton test case the potential function follows from the parameters we choose for our soliton (discrete eigenvalue). For this potential function the discrete eigenvalues are known analytically so we will also compute those. The derivation of the corresponding potential function is given in [20, Sec. 1]. Note that the authors give the definition of $\mathbf{q}^*$ and that their definition of x and t is swapped w.r.t. this report. For the definition of x and t we use in this report $\mathbf{q}(x,t)$ is given by:

$$\mathbf{q} = \operatorname{conj}\left(-2\eta\mathbf{c}\exp(2j\xi t + 4j(\xi^2 - \eta^2)x) \times \operatorname{sech}(2\eta(t - x_0) + 8\xi\eta x)\right). \tag{6-7}$$

The parameter ξ defines the velocity of the soliton as $v = -4\xi$ and η defines the amplitude. Parameters x_0 and $\mathbf{c}$ are both defined once we choose S, the vector defining the polarization of both elements of the signal:

$$\mathbf{c} = S/|S|, \qquad x_0 = \frac{1}{4\eta}\ln(|S|)^2. \tag{6-8}$$

The parameter S can be chosen freely. The variable x is the space coordinate where we determine the scattering transform of $\mathbf{q}(x,t) = \mathbf{q}(t)$. We can choose this freely as well. However, this x influences $\mathbf{q}(x,t)$. It therefore influences the T domain we should choose for the numerical implementation as this domain should be large enough to capture all dynamics of the signal. For these parameters, the location of the discrete eigenvalue is given by $\zeta = \xi + i\eta$ [20, Eq. 14] and the a coefficient in given by [20, Eq. 15]

$$a(\lambda) = \frac{\lambda - \zeta}{\lambda - \zeta^*}. \tag{6-9}$$

The coefficient a is the upper left element of the scattering matrix, and the remaining elements of this matrix vanish for all real λ: $b_{1,2} = 0$, as expected, as the spectrum of this $\mathbf{q}$ is a pure soliton and thus the continuous spectrum is zero.

This test case is interesting because it allows us to check if the discrete eigenvalues are determined correctly by the FNFT library. The test case we implemented has the following parameters:

Single soliton test case :
$T = [-31.5, 32.5], \quad \lambda \in [-110, 120],$
$\xi = 4.87, \quad \eta = 0.56, \quad x = 0.1, \quad S = [6, \quad 1 + 5i]$

The time domain T indicates the time domain where the samples of $\mathbf{q}(t)$ are taken and thus where we truncate $\mathbf{q}(t)$. This choice is motivated by the machine precision of FNFT_REAL, the datatype used for real numbers in the software library. Double precision is used, which implies that the machine precision is $\epsilon = 2^{-52} \approx 2.22 \times 10^{-16}$. This choice of T ensures that $|\mathbf{q}(t)| \leq \epsilon$ for $t \notin T$. For the region in the frequency domain we looked at the imaginary part of the $a(\lambda)$ coefficient (the absolute value is 1 everywhere). The maximum absolute value of the imaginary part of the $a(\lambda)$ coefficient is 1, so we again somewhat arbitrarily define the interesting region to be where the imaginary part of $a(\lambda) < 1/100 = 0.01$. That is true for $\lambda \in [-108, 117]$ which we rounded to $\lambda \in [-110, 120]$.

6-1-3 Double soliton

As with the single soliton, the potential function for the two-soliton case follows directly from the parameters we choose for the two solitons. For the expressions of the potential function we refer the reader to [34, Eq. 3.1 - 3.2]. Note that in this article the roles of x and t are swapped w.r.t. this thesis.

The expression for $\mathbf{q}(x,t)$ is given by

$$\mathbf{q}(x,t) = \frac{2}{\det(U)}\Bigg\{ \left(\frac{\exp(\tau_2+\alpha_2)}{2\eta_2} + \Sigma_{l=1}^{2}(\lambda_{22l}-\lambda_{21l})\exp\left(-(\tau_l+\alpha_l)+j(\theta_l-\theta_2)\right)\right)\exp(-j\theta_1)\mathbf{u_1} + \left(\frac{\exp(\tau_1+\alpha_1)}{2\eta_1} + \Sigma_{l=1}^{2}(\lambda_{11l}-\lambda_{12l})\exp\left(-(\tau_l+\alpha_l)+j(\theta_l-\theta_1)\right)\right)\exp(-j\theta_2)\mathbf{u_2}\Bigg\}, \tag{6-10}$$

where

$$\begin{aligned}
U &= \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \\
\text{with } U_{11} &= \frac{\exp(\tau_1+\alpha_1)}{2\eta_1} + \Sigma_{l=1}^{2}\lambda_{11l}\exp\left(-(\tau_l+\alpha_l)+j(\theta_l-\theta_1)\right), \\
U_{12} &= \Sigma_{l=1}^{2}\lambda_{12l}\exp\left(-(\tau_l+\alpha_l)+j(\theta_l-\theta_1)\right), \\
U_{21} &= \Sigma_{l=1}^{2}\lambda_{21l}\exp\left(-(\tau_l+\alpha_l)+j(\theta_l-\theta_2)\right), \\
U_{2}2 &= \frac{\exp(\tau_2+\alpha_2)}{2\eta_2} + \Sigma_{l=1}^{2}\lambda_{22l}\exp\left(-(\tau_l+\alpha_l)+j(\theta_l-\theta_2)\right), \\
\lambda_{jkl} &= -\frac{2\eta_l(\mathbf{u_j}\cdot\mathbf{u_l}^*)}{(\zeta_l-\zeta_k^*)(\zeta_l-\zeta_j^*)}, \\
\tau_j &= 2\eta_j(t+4\xi_j x), \\
\theta_j &= 2\xi_j t + 4(\xi_j^2-\eta_j^2)x, \\
\zeta_j &= \xi_j + j\eta_j.
\end{aligned}$$

Here $\mathbf{u_1}^*$ is the complex conjugate of vector $\mathbf{u_1}$. We implemented a testcase with the following parameters:

Double soliton test case:
$T = [-31.5, 30], \quad \lambda \in [-350, 350],$
$\xi_1 = 4.87, \xi_2 = 0.358, \quad \eta_1 = 0.56, \eta_2 = 1.28 \quad \alpha_1 = 1.5,$
$\alpha_2 = 3.6, \quad \mathbf{u}_1 = [j, 2]^T, \mathbf{u}_2 = [6, 2.2+3j]^T, x = 0.1$

Parameters $\xi_{1,2}$ and $\eta_{1,2}$ define the velocity and amplitude of the two solitons, $\alpha_{1,2}$ are used for a parametrization and $\mathbf{u}_{1,2}$ are unit vectors defining the polarizations of the signal. The domain T is chosen such that $\mathbf{q}_{1,2}(t) \leq \epsilon$ for $t \notin T$, where ϵ is the machine precision of real numbers in the software library. For the region of λ we chose values for which the real part of $a(\lambda) < 0.99$. For λ values outside this region the value barely changes and gets asymptotically closer to 1.

6-1-4 Secant hyperbolic

The potential function is given by

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} A_1\text{sech}(t) \\ A_2\text{sech}(t) \end{bmatrix}, \tag{6-11}$$

where $\text{sech}(t) = \frac{1}{\cosh(t)}$ is the secant hyperbolic. The NFT coefficients for the Nonlinear Schrödinger Equation (NSE) with secant hyperbolic potential $q = A\text{sech}(t)$ are derived in [30, Sec. 5]. The resulting expressions are

$$\begin{aligned} a &= \frac{\Gamma\left(\frac{1}{2} - j\lambda\right)^2}{\Gamma\left(\frac{1}{2} - j\lambda + A\right)\Gamma\left(\frac{1}{2} - j\lambda - A\right)} \\ b &= \frac{j\Gamma\left(\frac{1}{2} + j\lambda\right)^2}{\Gamma(A)\Gamma(1-A)} = j\frac{\sin(\pi A)}{\cosh(\pi\lambda)}, \end{aligned} \tag{6-12}$$

where $\Gamma(\cdot)$ is the gamma function. Note that the authors of [30] restricted themselves to only the focusing case and hence their expressions do not involve a dependence on κ. As far as we are aware, exact expressions for NFT coefficients for the Manakov equation have not been found yet. Deriving these results is non-trivial but we found the expressions for the coefficients by trial and error. We state them here and give reasons that support this result afterwards:

$$\begin{aligned} a &= \frac{\Gamma(\frac{1}{2} - \lambda j)^2}{\Gamma(\frac{1}{2} - \lambda j + q_0)\Gamma(\frac{1}{2} - \lambda j - q_0)}, \\ b_1 &= -\kappa\frac{\sin(\pi q_0)A_1\text{sech}(\lambda\pi)}{q_0}, \\ b_2 &= -\kappa\frac{\sin(\pi q_0)A_2\text{sech}(\lambda\pi)}{q_0}, \end{aligned} \tag{6-13}$$

where we defined $q_0 = \sqrt{\kappa}\sqrt{|A_1|^2 + |A_2|^2}$. Comparing 6-13 with 6-12, we see some similarities. The amplitude A from the NSE case is replaced by $q_0 = \sqrt{\kappa}\sqrt{|A_1|^2 + |A_2|^2}$. This makes sense, as both Eq. (2-10) and the NSE equivalent can be rewritten as second order systems for this potential function, and in that case $\sqrt{\kappa}\sqrt{|A_1|^2 + |A_2|^2}$ takes the place of A for the Manakov Equation (ME) case. For the b coefficients, note that $\frac{1}{\cosh(x)} = \text{sech}(x)$ and thus these expressions are similar as well. The b coefficient then also has another scaling term which has no direct analogue in the NSE solution.

We feel confident in assuming these expressions are correct for the following reasons: for the rectangle, single soliton and double soliton potential the nonlinear spectrum can be computed analytically. We can thus be sure that these are correct, and in Section 6-2 we will see that the results of all implemented numerical methods converge to these exact solutions. This implies that the methods have been implemented correctly. Secondly, we note that the potential function for the single soliton is actually a scaled secant hyperbolic, which implies that methods determining the NFT spectrum of the single soliton correctly will also do so for the secant hyperbolic. As we will see, all implemented the methods converge to this "trial and error solution" for the secant hyperbolic test case. We furthermore see that the order of error decay is in line with what we expect from second and fourth order methods. Thus, we can be quite sure these expressions are correct.

We did multiple tests with the secant hyperbolic potential with the following parameters:

Secant hyperbolic test case:

$A_1 = 0.8, \quad A_2 = 5.2, \quad T = [-38.5, \quad 38.5], \quad \lambda \in [-4.6, \quad 4.6]$

The domain T is again chosen to minimize the truncation error of $\mathbf{q}(t)$. The a coefficient tends to 1 as $\lambda \to \pm\infty$, while the $b_{1,2}$ coefficients tend to 0. We define the region of interesting dynamics as $\rho_{1,2} = b_{1,2}/a \leq 10^{-6}$. This is the case for $\lambda \leq -4.6$ and $\lambda \geq 4.6$, so we choose $\lambda \in [-4.6, 4.6]$.

6-1-5 A note on timestep size

We should note that the maximum and minumum λ where we can evaluate the NFT coefficients is determined by the timestep size h.

For the linear Fourier Transform (FT), one can use the Nyquist-Shannon sampling theorem [31] to determine the maximum value of h allowed to sample a signal with frequency content ω. Let ω be the highest frequency ([rad/s]) for which we want to determine the FT. Then we need to sample at a frequency of at least $2|\omega|$ [rad/s] $= \frac{2|\omega|}{2\pi}$[Hz] $= \frac{|\omega|}{\pi}$[Hz], which means $h \leq \frac{\pi}{|\omega|}$.

The authors of [23, p. 5] derived a similar limit on h for the NFT: $4h \leq \frac{2\pi}{\omega_{max}} \Rightarrow h \leq \frac{\pi}{2\omega_{max}}$, where $\omega_{max} = \sqrt{\lambda + |q_{\max}(t)|^2}$ is the maximum local frequency of the system Eq. (2-6) for a fixed λ. In this expression $|q_{\max}(t)| = \max_t |q(t)|$, the maximum of the absolute value of the potential $q(t)$.

However, the NFT tends to the linear FT in the limit case $||q|| << 1$ where 2λ from the NFT corresponds to ω from the linear FT [44, Sec. IV B]. For the NFT we will therefore use the Nyquist-Shannon sampling theorem to determine suitable sampling times as well. In the results discussed in Section 6-2, especially in the results for the secant hyperbolic test case, we will see that this indeed is an adequate bound on the sampling time. The sampling time for each test case should thus be at most $h \leq \frac{\pi}{|\omega|} = \frac{\pi}{|2\lambda|}$, where λ is the maximum value in the spectral domain for which we determine the NFT. For the rectangle test case we have $h \leq \frac{\pi}{2|250|} = \frac{\pi}{500} = 0.0063$. For the chosen time interval this means that the amount of samples in the time domain should be at least $\frac{T_2 - T_1}{0.0063} = 637$. For the single soliton we get $h \leq 0.0131$ and $D \geq 4890$, for the double soliton $h \leq 0.0045$ and $D \geq 1371$ and for the secant hyperbolic $h \leq 0.3415$ and $D \geq 226$. For lower D values the algorithms will not be able to determine the NFT correctly.

6-2 Results and discussion

For the performance metric of the tests, we used the relative L^2 error:

$$E_a = \frac{\sqrt{\sum_{k=1}^{M} |a_{\text{exact}}(\lambda_k) - a_{\text{num}}(\lambda_k)|^2}}{\sqrt{\sum_{k=1}^{M} |a_{\text{exact}}(\lambda_k)|^2}}, \tag{6-14}$$

where the a coefficient can also be replaced by $b_{1,2}$ or $\rho_{1,2}$. The λ_k's are M equidistant points in the λ-interval. In the subsequent sections, we will mostly analyze the results for computing the a coefficient. Results for the $b_{1,2}$ coefficients are mostly the same in qualitative terms. For $\rho = b/a$, numerical errors may be amplified for small a values. This effect is a bit less predictable than other numerical errors so the results of ρ might be a bit harder to analyze.

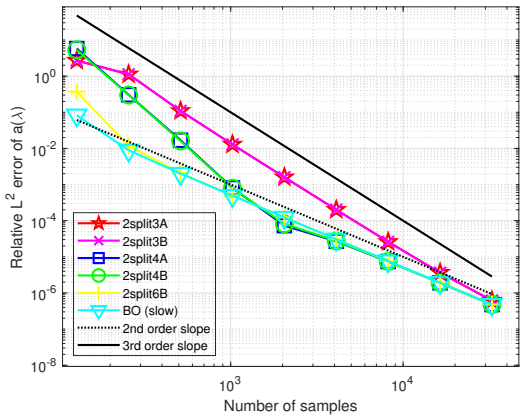
The number of samples in the frequency domain M is equal to the number of samples in the time domain D for all tests. Choosing $M = D$ is common practice in the field of communications and in Section 3-2 and Section 4-9 we determined the computational complexity orders for this case.

All tests were done for the focusing Manakov equation unless stated otherwise.

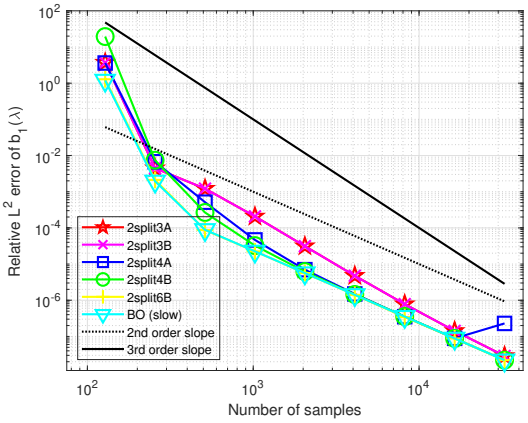
For all tests the code implemented in the library has been used. For the runtimes, background processes on the machine can have an influence and slight variations will occur for each run. The absolute runtimes of course heavily depend on the machine the tests are run on and we should therefor only look at the results of methods relative to each other. Each method has been run three times and the runtimes were averaged to minimize the effect of the slight fluctuations in run. In the following subsections we show and discuss the results for each of the test cases.

6-2-1 Secant hyperbolic test case

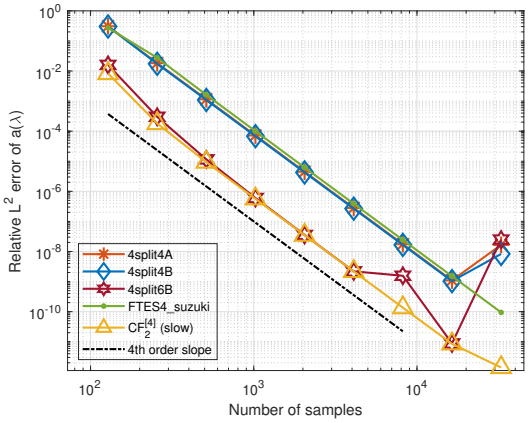
We show and analyse most results for the secant hyperbolic because this potential function both has an interesting continuous spectrum, unlike the single and double soliton test cases where the continuous spectrum is zero, and there are no jumps in the potential function as with the rectangle test case. Figure 6-1 shows the error in a and b_1 coefficients for all numerical results. In this and all other test cases, the relative errors for b_2 were virtually the same as for b_1. We therefore show only the results for b_1.



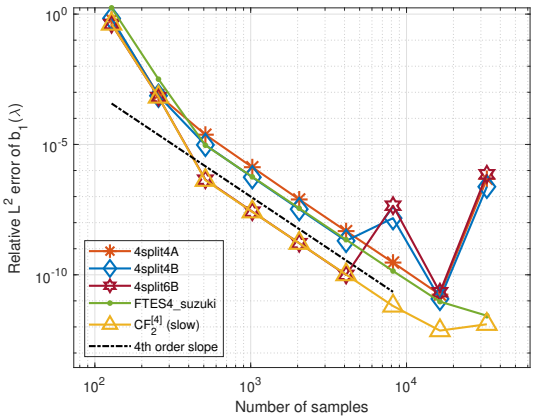
(a) Errors in a for second-order methods



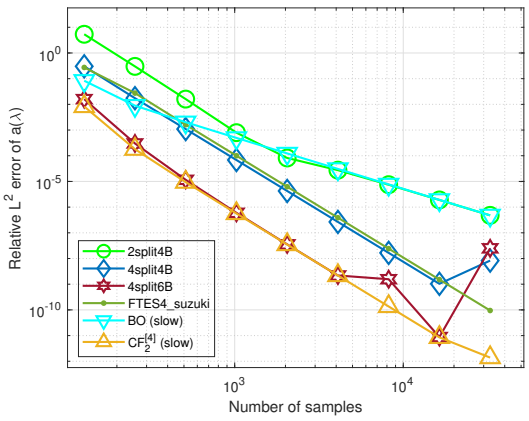
(b) Errors in b_1 for second-order methods



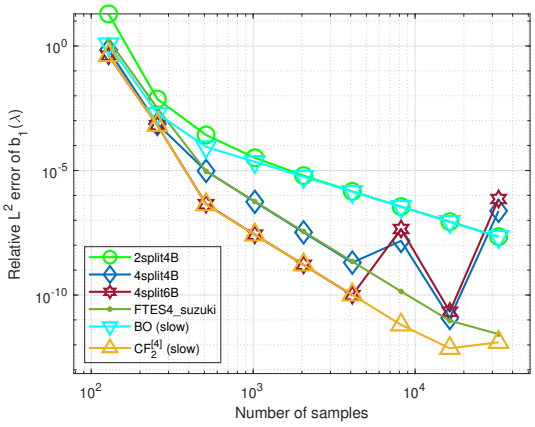
(c) Errors in a for fourth-order methods



(d) Errors in b_1 for fourth-order methods

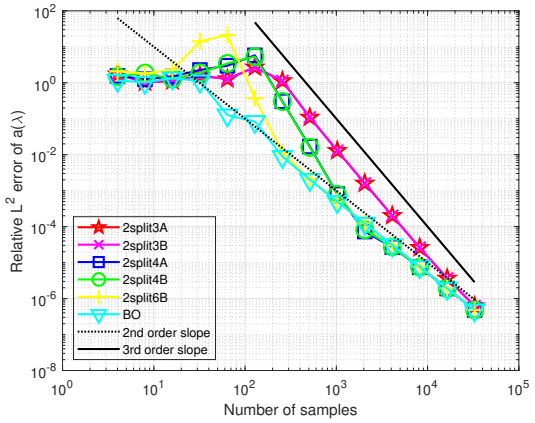


(e) Summary of errors in a

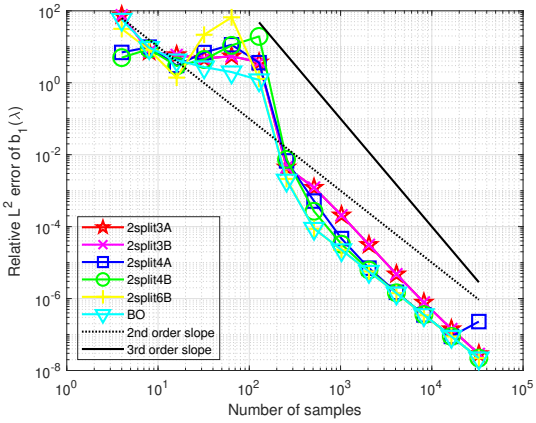


(f) Summary of errors in b_1

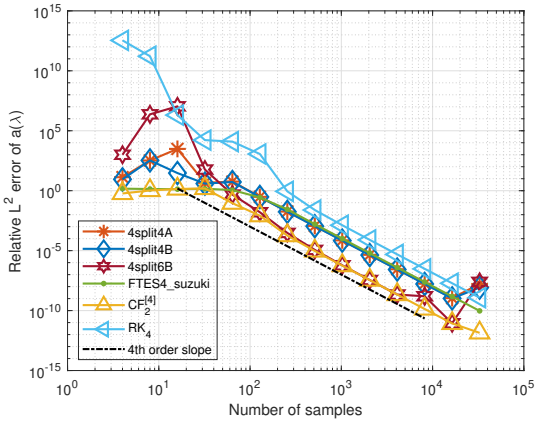
Figure 6-1: Errors against number of samples for computing the NFT of the focusing Manakov equation using slow and fast methods, secant hyperbolic potential function as given in Section 6-1-4 and $M = D$



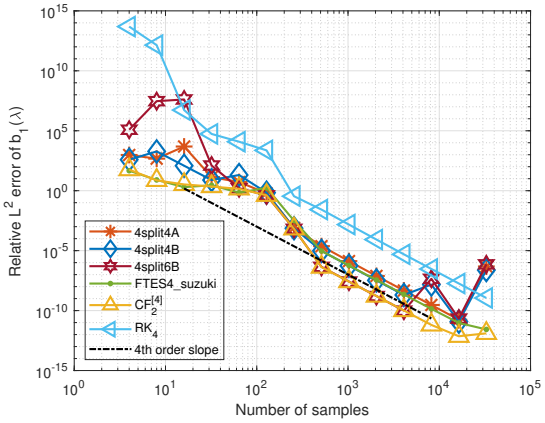
(a) Errors in a for second-order methods



(b) Errors in $b1$ for second-order methods



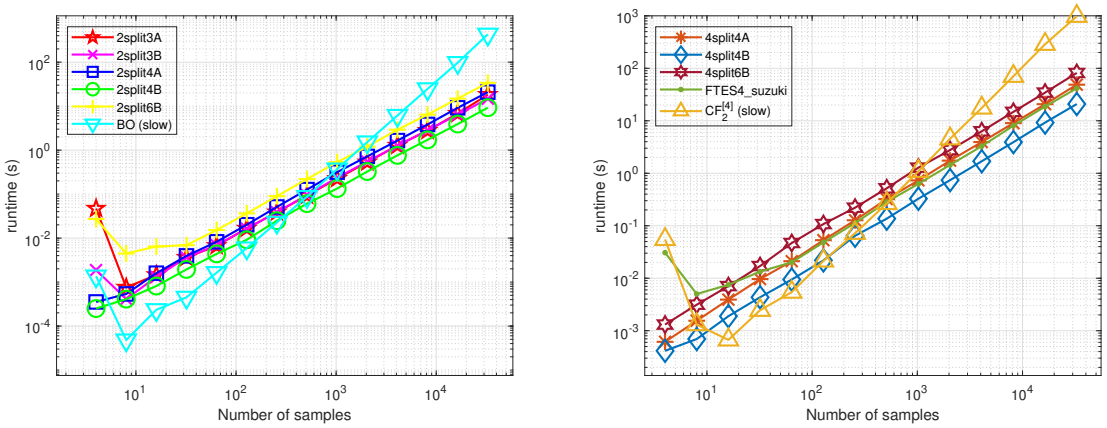
(c) Errors in a for fourth-order methods



(d) Errors in b_1 for fourth-order methods

Figure 6-2: Errors against number of samples for computing the NFT of the focusing Manakov equation using slow and fast methods, secant hyperbolic potential function as given in Section 6-1-4 and $M = D$. Result of choosing the sampling time too large is visible for low numbers of samples.

In Figure 6-1a we observe something interesting: the order of error decay of the fast methods is higher than second-order for lower numbers of samples. The methods 2split3A/B even display higher than second-order error decay for all numbers of samples. This is most likely because the splitting error instead of the discretization error is dominant in this region; we see that the error decay for 2split3A/B is third-order, and for 2split4A/B and 2split6B it is fourth and sixth-order respectively before becoming second-order. For the b_1 coefficient, we observe similar results, but the initial period where the splitting is the dominant error is smaller, so small even that we do not see it for 2split4A/B and 2split6B. In these cases we observe only the second-order discretization error, provided the number of time samples $D \geq 226$ as dictated by the Nyquist-Shannon sampling theorem. The performance results are as expected: the fast methods display slightly higher errors than BO because of the error introduced by splitting, and higher order splittings introduce a smaller error.
The fourth-order methods in Figure 6-1c and Figure 6-1d do display fourth-order error decay for the most part for the region $D \geq 226$. For the fast and $\mathrm{CF}_2^{[4]}$ methods we see that the error of both coefficients saturates or even becomes higher again for the highest numbers of samples. This is the error from truncating the potential function signal at $\mathbf{q}(x_0, t) \leq \epsilon$ where ϵ is the precision of real numbers in the library. If we want to eliminate this error we could increase the support T. We see again that the fast methods perform slightly worse due to the splitting error. We included the RK4 method for reference but notice that it performs worse than even the fast methods. Figure 6-1e and Figure 6-1f have been added for convenience to compare the methods that perform best in terms of error against number of samples.
The result of choosing the number of samples too low is seen most easily in the zoomed-out plots for the b coefficient that we show in Figure 6-2b and Figure 6-2d. We see that indeed the the error decreases sharply between 128 and 256 samples. In the previous section we determined that the number of samples should be at least $D = 226$ according to the Nyguist-Shannon sampling theorem.



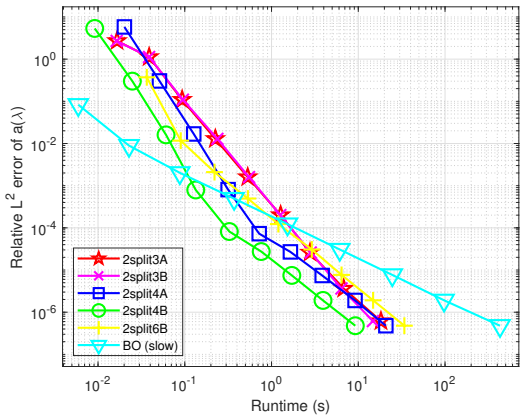
(a) Runtime second-order methods

(b) Runtime fourth-order methods

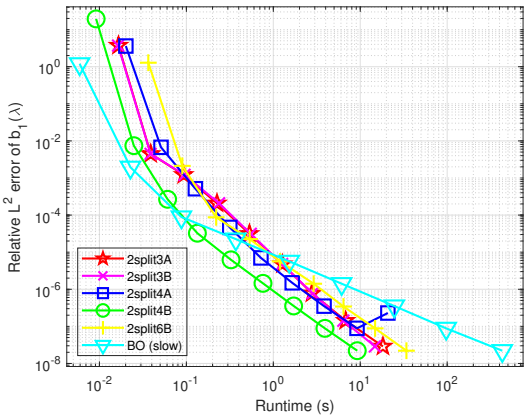
Figure 6-3: Runtime against number of samples for computing the NFT of the focusing Manakov equation using slow and fast methods, secant hyperbolic potential function with parameters as given in Section 6-1-4 and $M = D$

In Figure 6-3 we show the runtime of all methods against the number of samples. We see that for lower numbers of samples, the fast methods have a higher runtime. This is because they are not faster in absolute terms, only asymptotically for higher numbers of time and frequency domain samples. This is indeed what we observe for higher numbers of samples. The exact crossover point depends on the specific method, but for all fast methods this crossover point is indeed achieved for the last sample $D \leq 32768$.

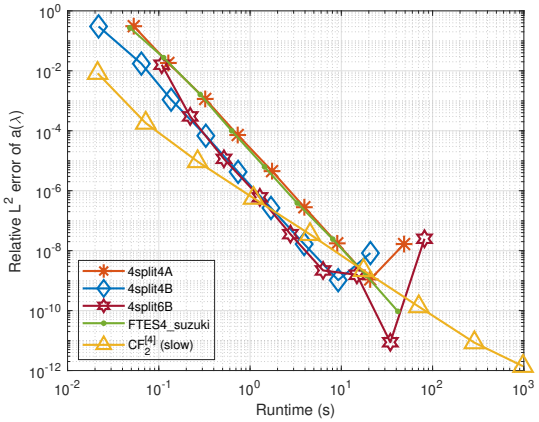
We should keep in mind however, that the fast methods also had a higher error for the same number of samples than the slow methods. We therefore plot the error against runtime in Figure 6-4. For the second-order methods, we see for both the a and b_1 coefficient in Figure 6-4a and Figure 6-4b that the BO method is a better choice for lower numbers of samples, albeit only slightly for the b_1 coefficient. For higher numbers of samples the fast methods perform better. For the fourth-order methods in Figure 6-4c and Figure 6-4d this effect is less clear; the error saturates due to the truncation error shortly after the fast methods take over the slow methods in terms of performance. The RK4 method has been omitted in these and other runtime plots as this method has not been implemented in the Fast Nonlinear Fourier Transform (FNFT) library. It was only implemented in MATLAB to generate results that could serve as a reference, and therefore looking at the runtime would not be a fair comparison. Again overview plots have been included in Figure 6-4e and Figure 6-4f.



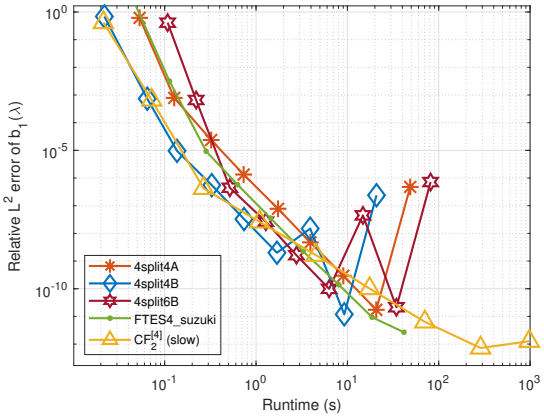
(a) Errors in a for second-order methods



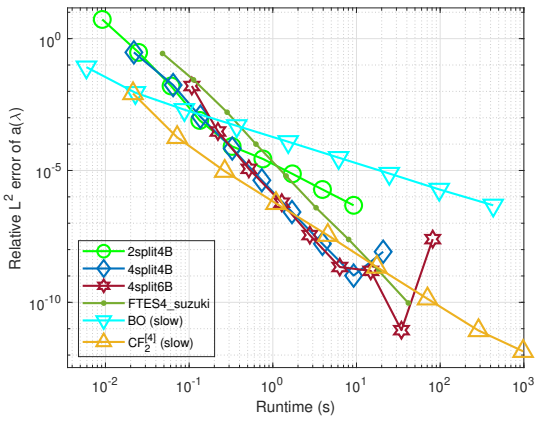
(b) Errors in b_1 for second-order methods



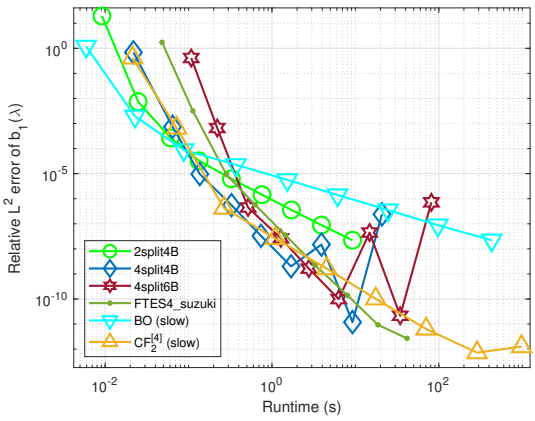
(c) Errors in a for fourth-order methods



(d) Errors in b_1 for fourth-order methods



(e) Summary of errors in a



(f) Summary of errors in b_1

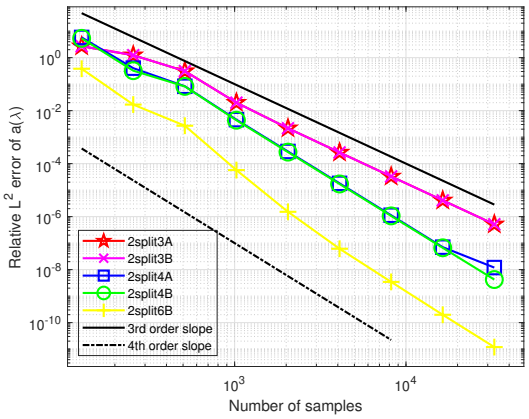
Figure 6-4: Errors against runtime for computing the NFT of the focusing Manakov equation using slow and fast methods, secant hyperbolic potential function with parameters as given in Section 6-1-4 and $M = D$

Results for the secant hyperbolic test case with Richardson Extrapolation

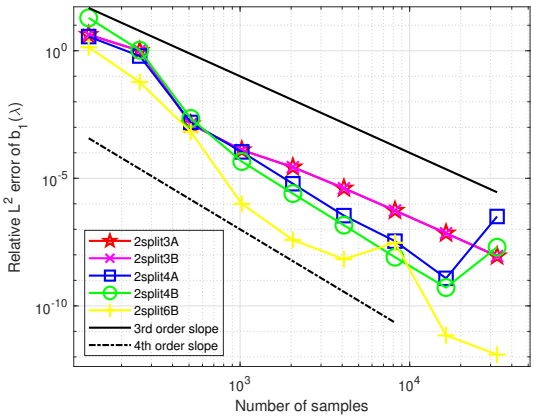
For the fast methods we also implemented the option to use Richardson Extrapolation in the library. See Section 4-10 for more details on how this is implemented. We show the errors plotted against the number of time samples in Figure 6-5. For the slow methods we did not include Richardson Extrapolation in the implementation which is why they are not included in the plots.

In theory, Richardson Extrapolation should increase the order of the methods by 1. For the second order methods we already concluded in the case without RE that the order was determined by the splitting method and not by the discretization error for methods 2split3A/B. RE does not influence this error so we do not expect to see the result in these plots. Indeed, Figure 6-5a and Figure 6-5b show that the order for these methods is not increased. For the 2split4A/B and 2split6b methods however, which did display second-order error decay for higher numbers of samples in the case without Richardson Extrapolation (RE), we do see that the order is increased due to RE. In Figure 6-5c we observe that the 4split4A/B methods now have fifth-order error decay. For 4split6B and FTES4 the error decay is even higher than the expected fifth-order behaviour. A similar effect was also observed in [10] for fast implementations of BO and $CF_2^{[4]}$ with Richardson Extrapolation applied to the NSE.

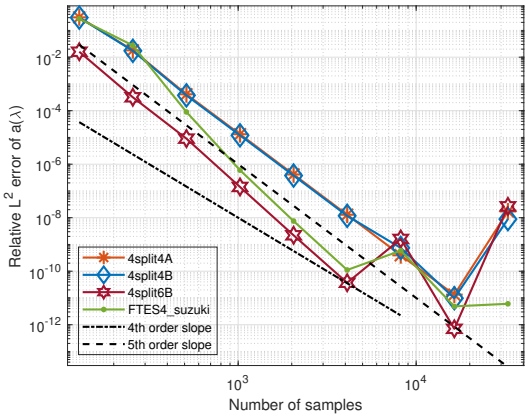
We show the results with and without Richardson extrapolation side by side in Figure 6-6 for easy comparison. The tests with and without RE both used the same numbers of time samples. For the 2split4A/B and 2split6B methods we clearly see that the error with RE is less for the higher numbers of time samples. For all these methods there is also a crossover point where the variation with RE gives a lower error for the same computation time. With 2split3A/B we do not see this effect as clearly. It is possible that we would see this effect with higher numbers of time samples.
In 4split4A/B and 4split6B we see the unpredictable behaviour at the end where the error goes up amplified with RE. This is also one of the dangers of RE: we need to know the order of error decay of the method for this to work. If the order of error decay is not the expected order, RE does not work and even gives worse results. We do however observe a slightly better runtime / error trade off for all methods in the region of $10^{-5} - 10^{-10}$ relative error.



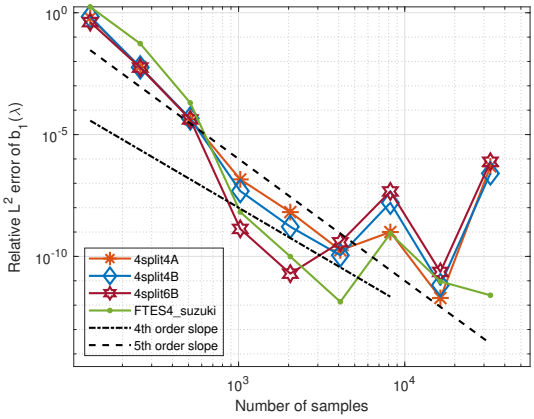
(a) Errors in a for second-order methods



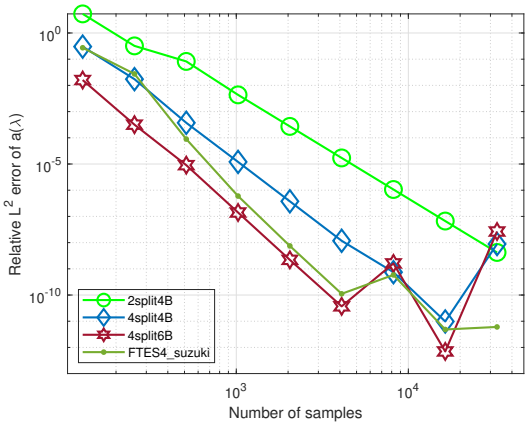
(b) Errors in b_1 for second-order methods



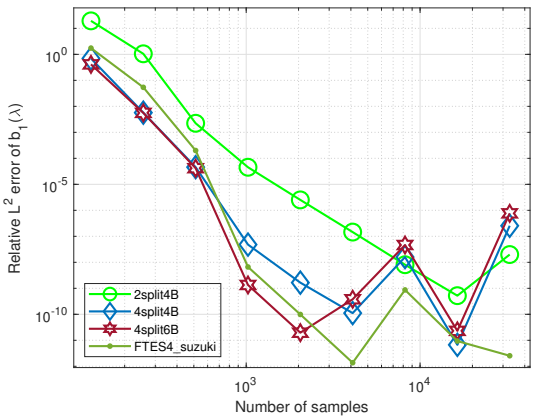
(c) Errors in a for fourth-order methods



(d) Errors in b_1 for fourth-order methods

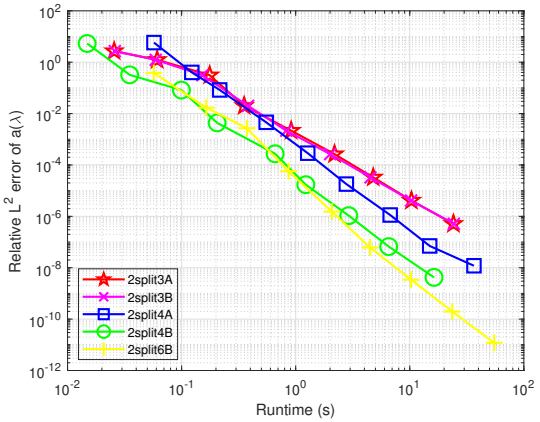


(e) Summary of errors in a

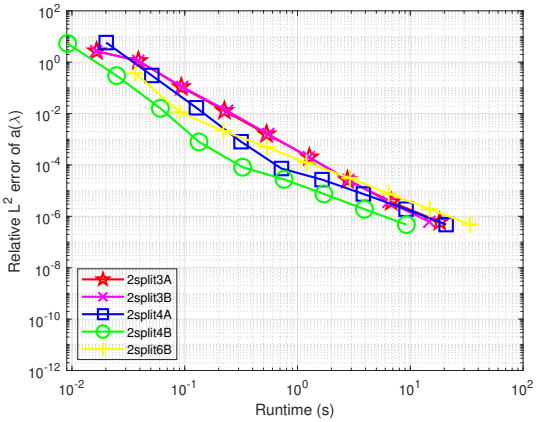


(f) Summary of errors in b_1

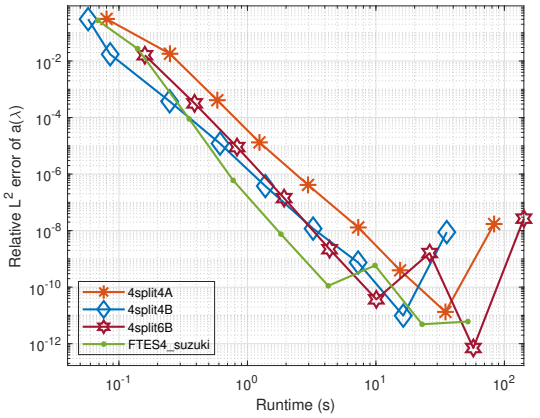
Figure 6-5: Errors against number of samples for computing the NFT of the focusing Manakov equation using fast methods with Richardson Extrapolation, secant hyperbolic potential function as given in Section 6-1-4 and $M = D$



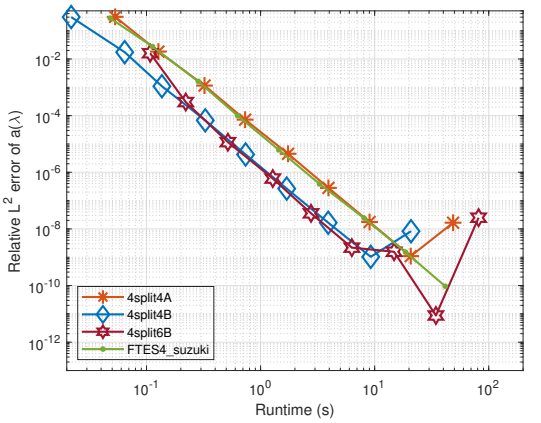
(a) Errors in a for second-order methods with RE



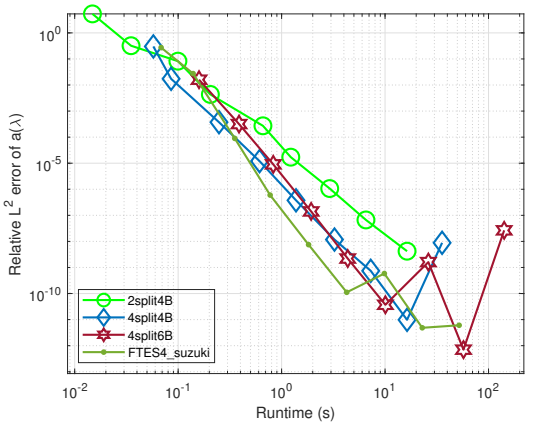
(b) Errors in a for second-order methods without RE



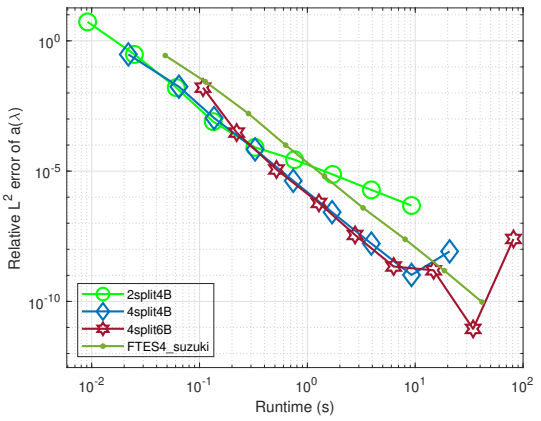
(c) Errors in a for fourth-order methods with RE



(d) Errors in a for fourth-order methods without RE



(e) Summary of errors in a with RE



(f) Summary of errors in a without RE

Figure 6-6: Errors against runtime for computing the a coefficient of the focusing Manakov equation using fast methods with and without Richardson Extrapolation, secant hyperbolic potential function as given in Section 6-1-4 and $M = D$

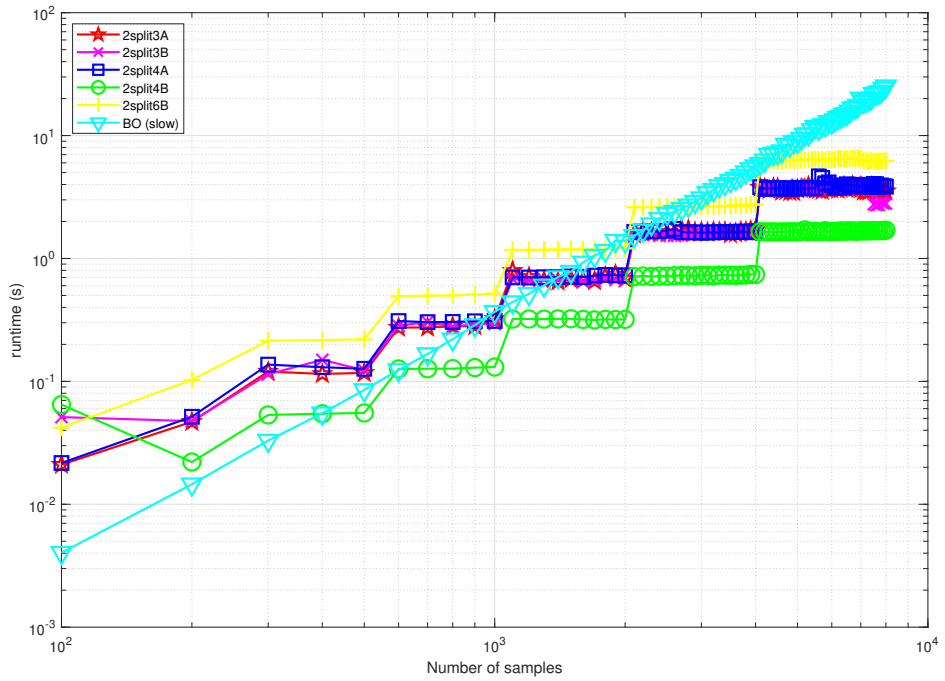
Results for the secant hyperbolic test case with different numbers of time samples

For all previous tests in this subsection and all tests concerning the other test cases we set the numbers of time samples to be a power of 2. As noted in Section 4-6 this leads to the most computationally efficient form of multiplication as the transition matrices to be multiplied are padded with identity matrices if the number of samples is not a power of 2. In Figure 6-7 we show the runtimes against numbers of samples where D is not a power of 2. For all fast methods we notice a kind of staircase figure. This is because of the padding in the multiplication step. The multiplication step is dominant for the runtime and the runtime for D samples will always be the same as the runtime for 2^f samples, where f is the lowest integer such that $D \leq 2^f$. For the slow methods this is not an issue and we see that the runtime increases with $\mathcal{O}(D^2)$. We remark on a possible solution to decrease the runtime of cases with D not a power of 2 in Section 7-3.

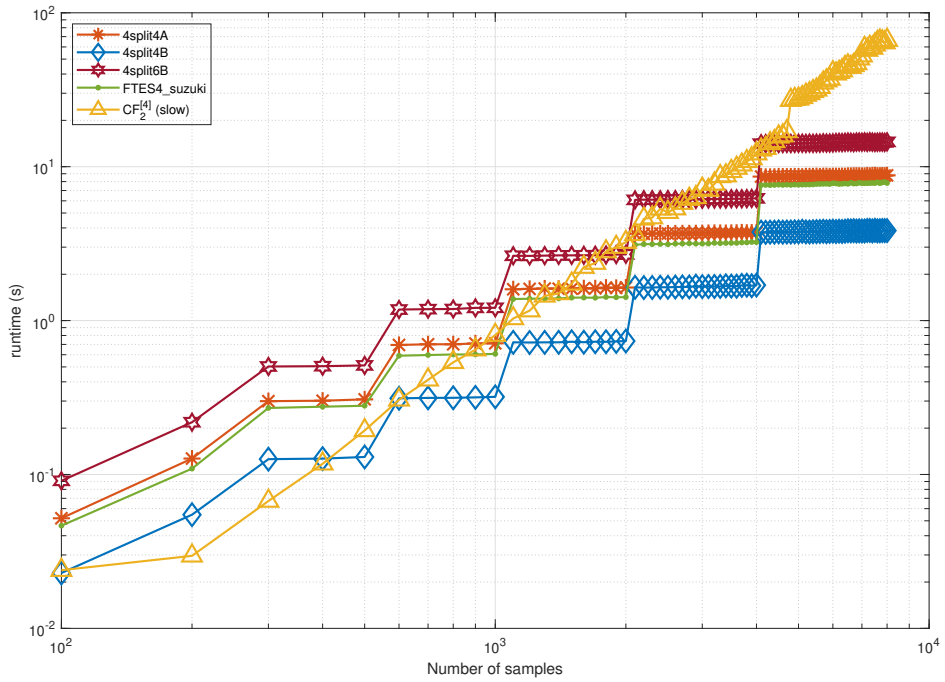
Results for the defocusing tests

The defocusing case is, in terms of the algorithm, not fundamentally different from the focusing case so these tests primarily serve as a check of the implementation of the code. We only did these tests for the secant hyperbolic and assume that performance results for the rectangle test case will be similar. We chose the same ξ and as this turned out to be the interesting region for the defocusing case as well where at the boundaries the value of a was reduced to around 0.01 of its maximum value at $\lambda = 0$. As the potential function $\mathbf{q}(x,t)$ is the same as for the focusing case, the T interval can stay the same as well. We plot the results in Figure 6-8.

In the plots for the second-order methods Figure 6-8a and Figure 6-8b we observe the same qualitative results as for the focusing case. For the fourth-order methods, there are some specific parts where some of the fast methods perform better, but as with the focusing the error saturates before the fast methods achieve a better performance than the slow methods.

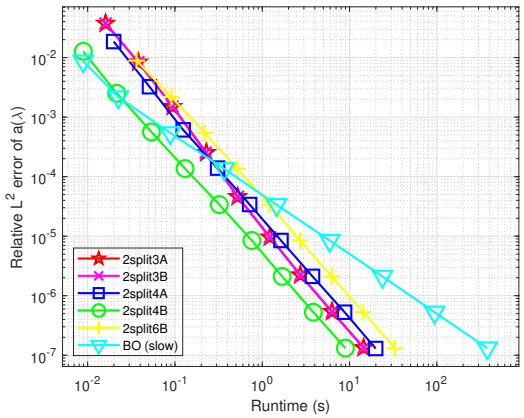


(a) Runtime against number of samples for the second-order methods

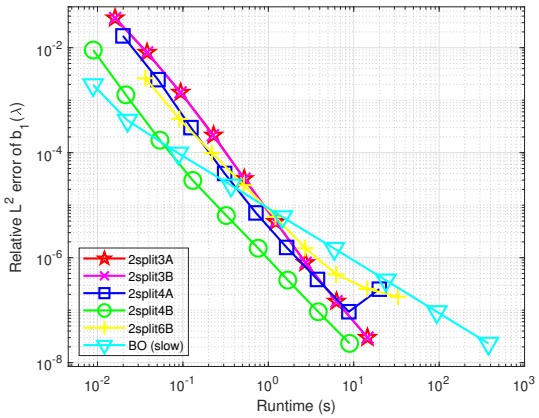


(b) Runtime against number of samples for the fourth-order methods

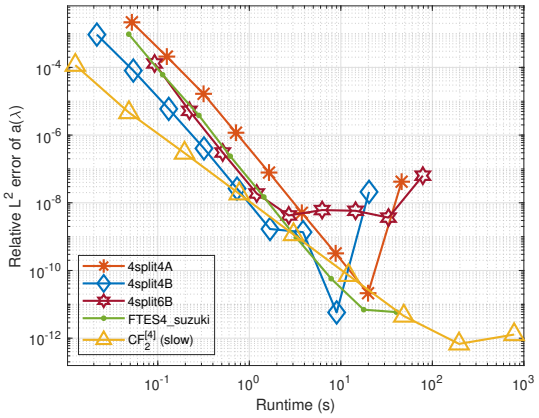
Figure 6-7: Runtime against samples for computing the NFT coefficient of the focusing Manakov equation using fast and slow methods, secant hyperbolic potential function as given in Section 6-1-4 and $M = D$ where D is not a power of 2



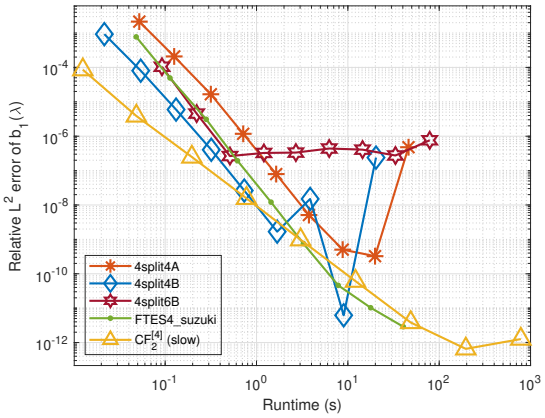
(a) Errors in a for the second-order methods



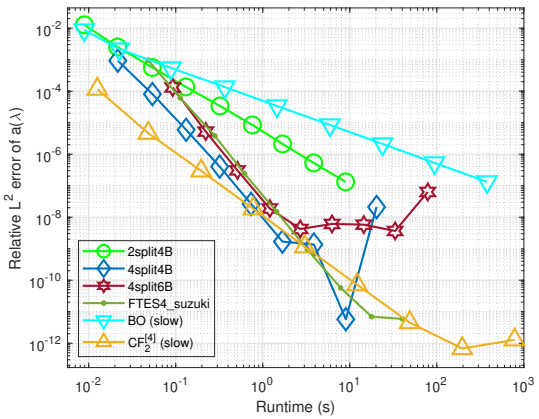
(b) Errors in b_1 for the second-order methods



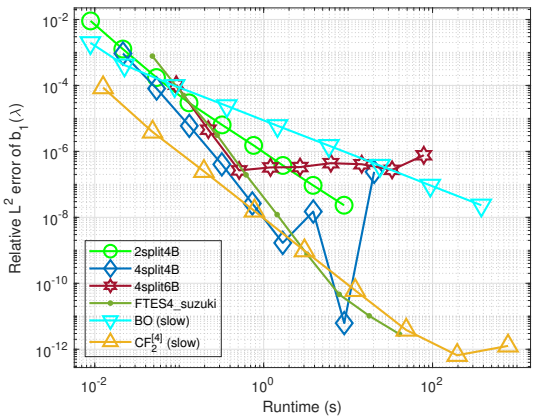
(c) Errors in a for the fourth-order methods



(d) Errors in b_1 for the fourth-order methods



(e) Errors in a for multiple methods



(f) Errors in b_1 for multiple methods

Figure 6-8: Error against runtime for computing the NFT coefficient of the defocusing Manakov equation using fast and slow methods, secant hyperbolic potential function as given in Section 6-1-4 and $M = D$

6-2-2 Rectangle test case

In the previous subsection we gave some results which show that the fast algorithms are indeed faster than the slow methods in specific cases. Having established that, we will focus here on error / runtime plots as they are quite informative for users of the software. Before that however we show an interesting error / samples plot for the second-order methods in Figure 6-9.

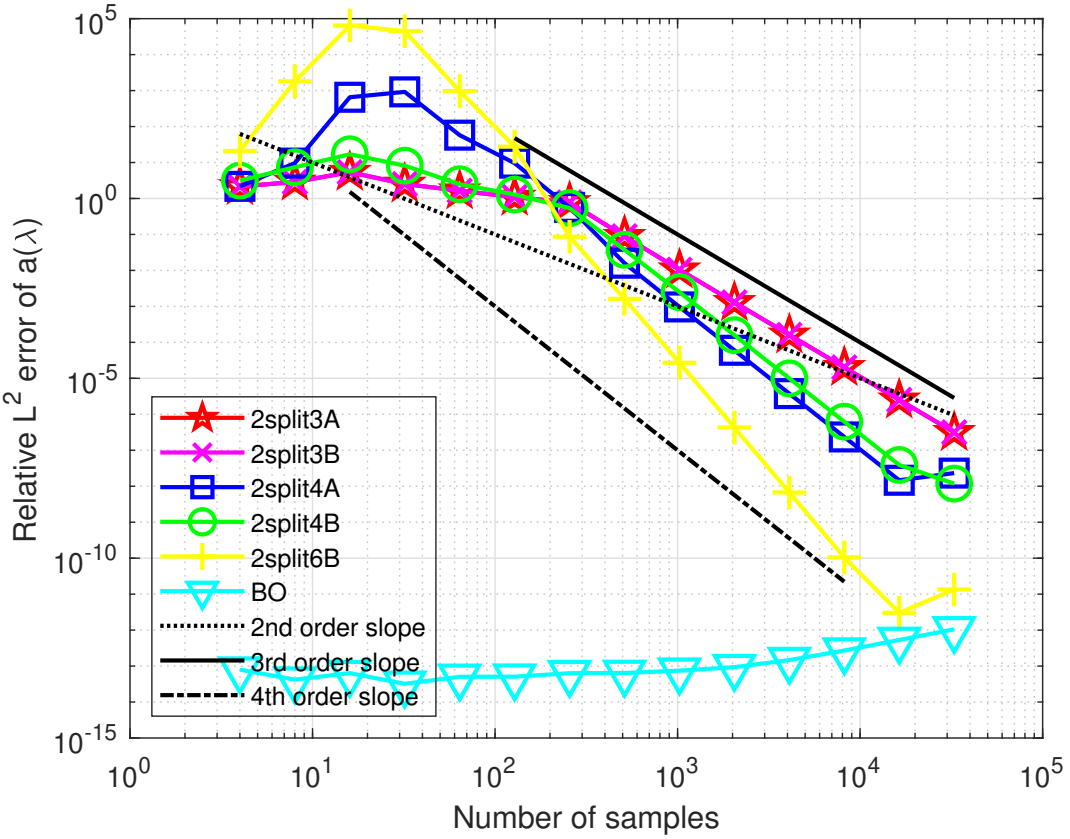
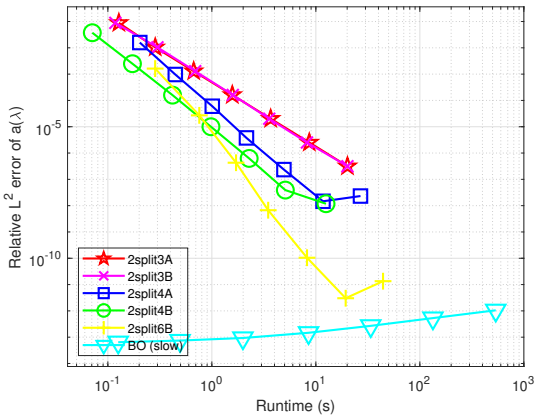


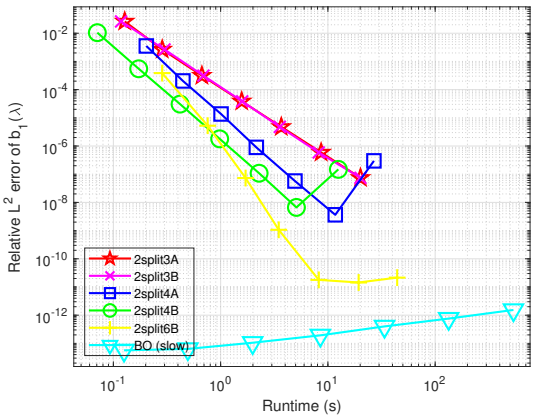
Figure 6-9: Error against the number of samples for computing the NFT of the focusing Manakov equation using fast and slow second-order methods, rectangle potential function as given in Section 6-1-1 and $M = D$

We notice that BO has a very low relative error. We recall from Section 3-1 that BO approximates the potential function in each timestep to be constant and then gives the exact solution. If we choose the time interval for the rectangle potential such that the jumps at $-L$ and L are at the boundaries of the $\mathbf{v}$ interval, the potential function is actually constant in over the whole interval. It is then also constant over each time step and thus BO determines the solution exactly. The small remaining error is due to rounding errors, which is why it grows with the number of samples: more samples mean more rounding errors.
In Figure 6-10 we plot the errors against the runtimes. Apart from the low error of BO the results in Figure 6-10a and Figure 6-10b are similar to the results from the secant hyperbolic test case, with the difference that for the rectangle potential the 2split6B method has the

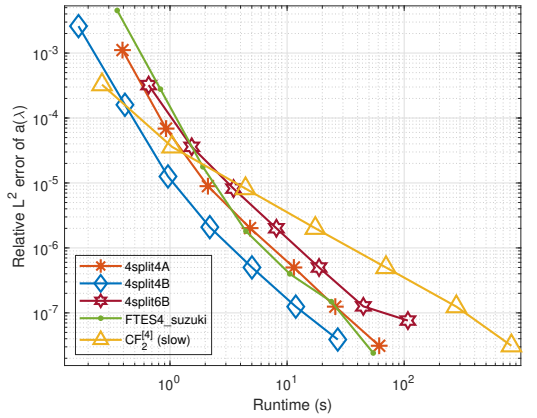
lowest error of the fast methods. For the fourth-order methods we observe that there is a crossover point where the fast methods start to outperform the $CF_2^{[4]}$ method. 4split4B offers the best error / runtime trade-off.



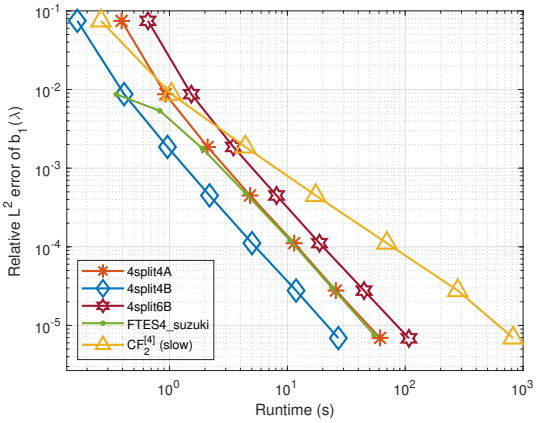
(a) Errors in a for second-order methods



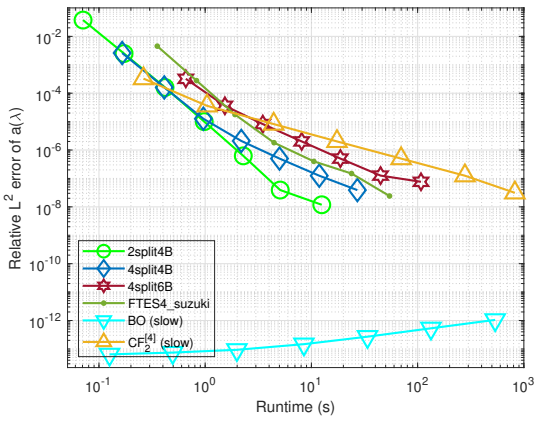
(b) Errors in b_1 for second-order methods



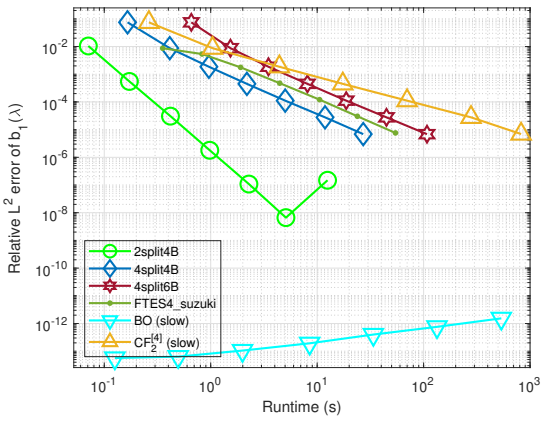
(c) Errors in a for fourth-order methods



(d) Errors in b_1 for fourth-order methods



(e) Summary of errors in a



(f) Summary of errors in b_1

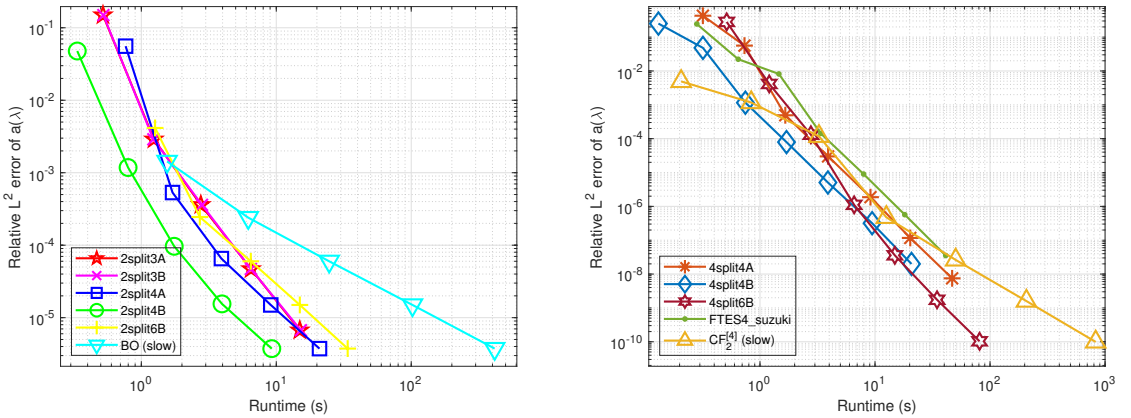
Figure 6-10: Errors against the runtime for computing the NFT of the focusing Manakov equation using fast and slow methods, rectangle potential function as given in Section 6-1-1 and $M = D$

6-2-3 Single soliton test case

The single and double soliton test cases are pure solitons and thus the continuous spectrum ρ and also the NFT coefficient b are zero. We therefore only focus on the a coefficient and the discrete spectrum in these test cases.

Results for determining the a coefficient for the single soliton test case

We show the error against runtime for computing the a coefficient in Figure 6-11. the quantitative results are similar to what we saw for the secant hyperbolic: compare for example Figure 6-11a to Figure 6-4a. This makes sense, as the soliton potential function is a special case of the secant hyperbolic. Comparing Figure 6-11b to Figure 6-4c however we see that in the single soliton test case the error does not saturate and we do see the crossover point where the fast methods start to outperform the slow methods.



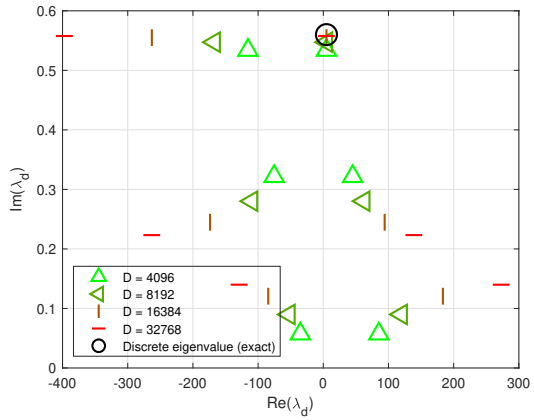
(a) Errors in a computed by second-order methods

(b) Errors in a computed by fourth-order methods

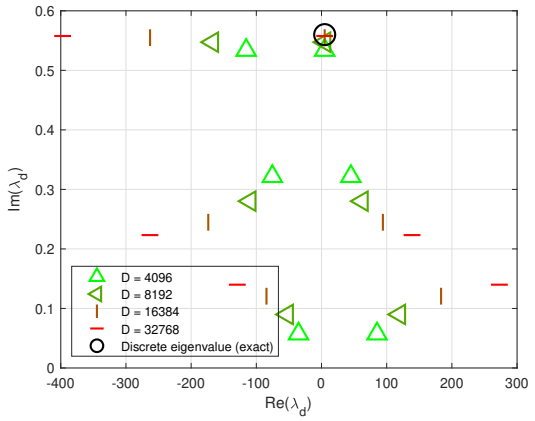
Figure 6-11: Errors against the runtime for computing the a coefficient of the focusing Manakov equation using fast and slow methods, single soliton potential function as given in Section 6-1-2 and $M = D$

Determining the discrete spectrum of the single soliton test case

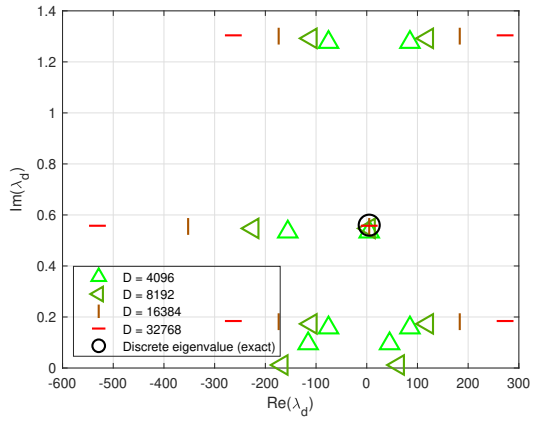
We determined the discrete eigenvalues (also known as bound states) of the test case given in Section 6-1-2 using the method from Section 4-8. The exact value of the single discrete eigenvalue is $\lambda_d = \xi + i\eta = 4.87 + 0.56i$. We plot the eigenvalues determined by the different fast methods for multiple numbers of time samples in Figure 6-12.



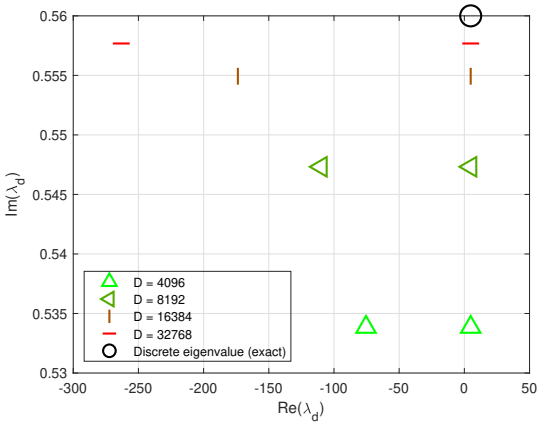
(a) 2split3A



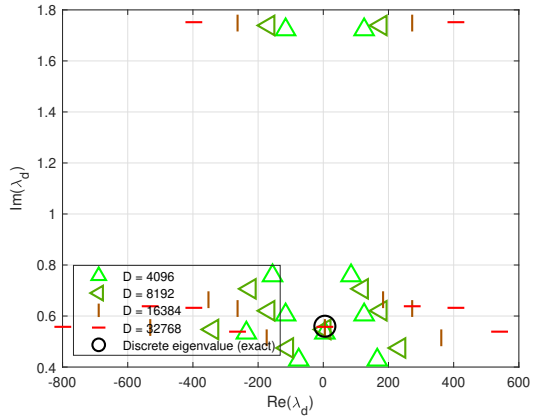
(b) 2split3B



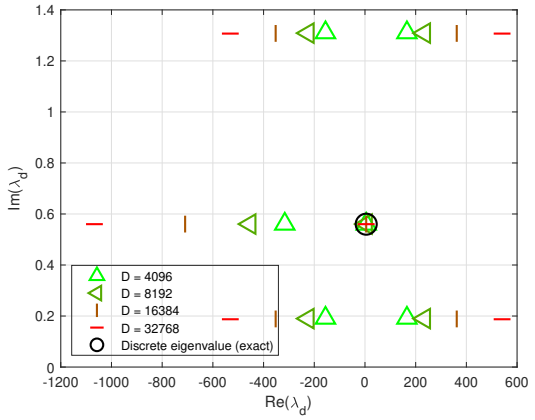
(c) 2split4A



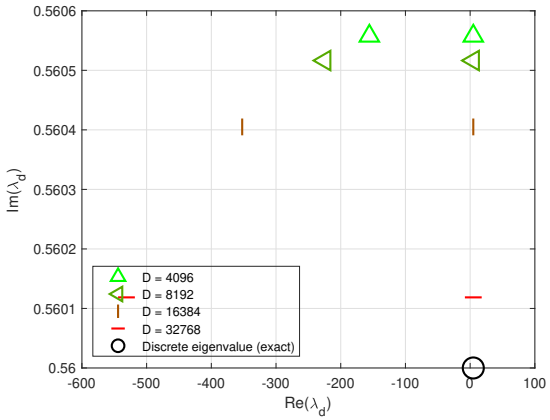
(d) 2split4B



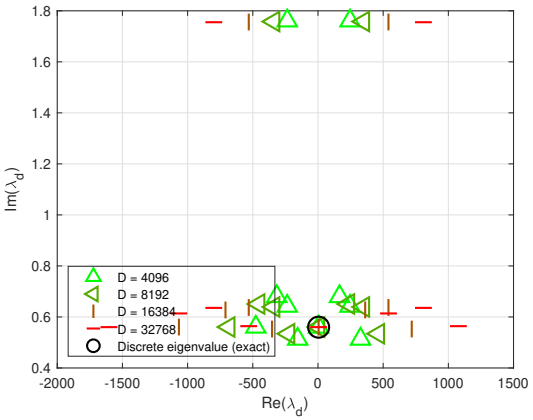
(e) 2split6B



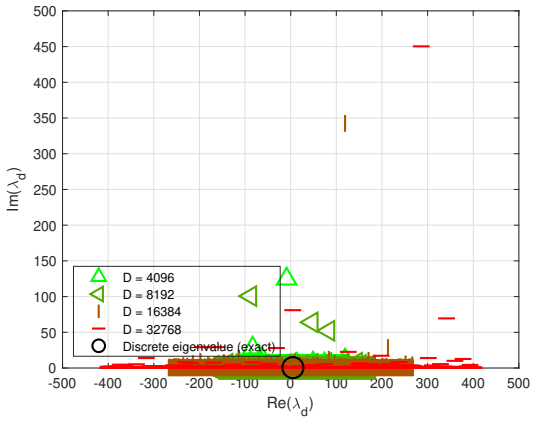
(f) 4split4A



(g) 4split4B



(h) 4split6B



(i) FTES4 with suzuki splitting

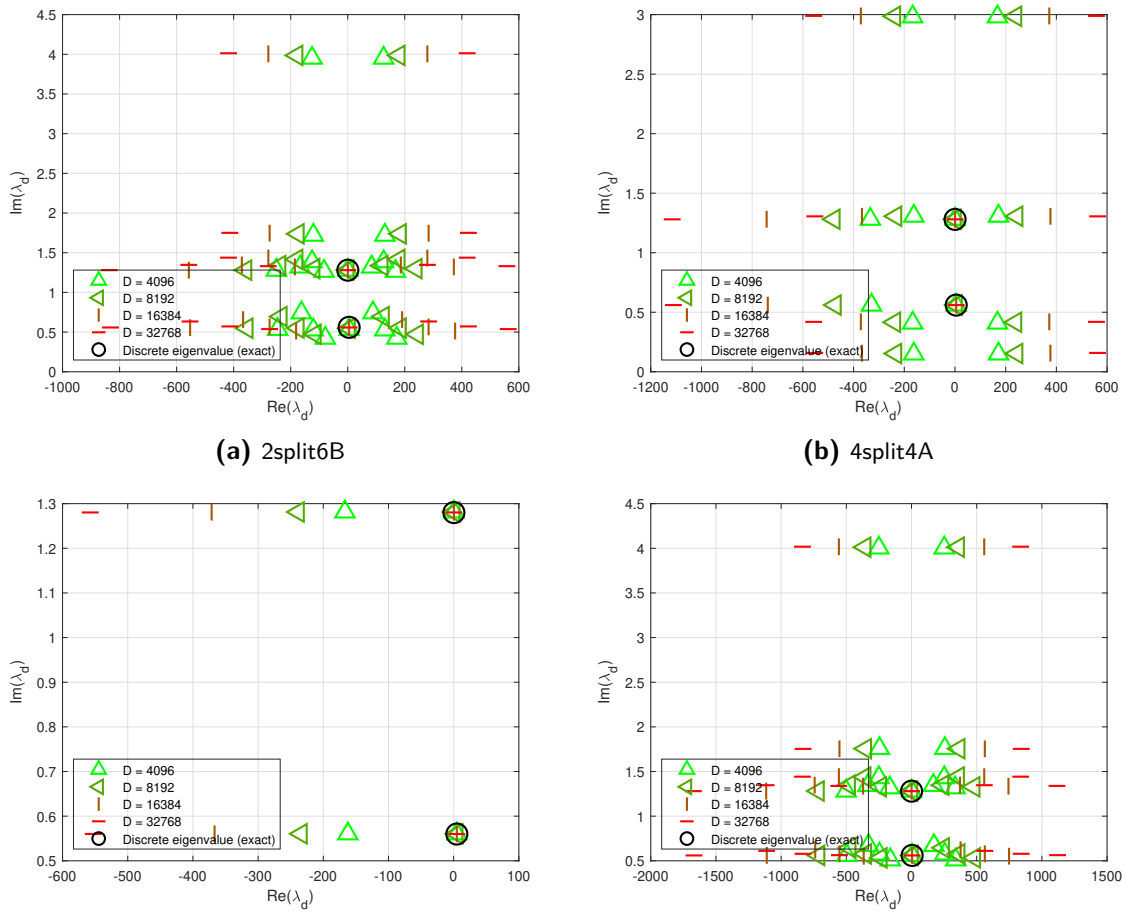
Figure 6-12: Numerical approximations of the bound states of the single soliton test case from Section 6-1-2 computed by the different fast methods for different numbers of time samples

For all methods except the FTES4 method the fast rootfinder determines the bound state correctly, with the error getting smaller as D increases. However, a lot of spurious values are found. This is to be expected as the roots of polynomial $\mathbf{G}_{11}$ numerically approximate the roots of $a(\lambda)$ instead of giving an exact expression. In [10, Appendix C] the authors also found spurious eigenvalues when computing the discrete eigenvalues for the NSE. They used the same approach as we used for the ME. Various filtering techniques can be devised to only select the true eigenvalue. In all 2split and 4split methods these spurious values have a large error in their real part that grows as D increases. So discarding all found values with a large absolute real part might be a good approach. However, if the exact solution is unknown, it might be difficult to determine exactly how large the real part should be before the value can be discarded.

6-2-4 Double soliton test case

For the test case in Section 6-1-3 the exact values of the bound states are given by $\lambda_{d1} = 4.87 + 0.56\text{i}, \lambda_{d2} = 0.358 + 1.28\text{i}$. The results for some of the fast methods are given in

Figure 6-13.



(a) 2split6B **(b)** 4split4A

(c) 4split4B **(d)** 4split6B

Figure 6-13: Numerical approximations of the bound states of the double soliton test case from Section 6-1-3 computed by the different fast methods for different numbers of time samples

We see the same qualitative results as for the single soliton: spurious eigenvalues with an increasingly large real part as D increases and the bound states are determined more accurately for higher numbers of samples.

Chapter 7

Conclusions and recommendations

7-1 Final results

We started out this thesis project with the following objectives:

In this thesis we aim to extend the first Fast Nonlinear Fourier Transform methods as proposed for the Nonlinear Schrödinger Equation to the dual-polarization case of the Manakov equation. We will develop these Fast Nonlinear Fourier Transform algorithms, integrate them in the existing open-source software library and benchmark them.

We developed multiple fast NFT algorithms for the Manakov equation: second-order methods based on the BO method with third, fourth and sixth-order exponential splitting, fourth-order methods based on $\text{CF}_2^{[4]}$ with fourth and sixth-order splitting and a fourth-order method based on TES4 with Suzuki factorization as the exponential splitting. The developed algorithms numerically determine the continuous nonlinear spectrum of the Manakov equation with a given potential function. We implemented these methods and integrated them in a C-based fast NFT software library, along with implementations of the BO and $\text{CF}_2^{[4]}$ methods which served as a tool for comparison. We also implemented the option to use Richardson extrapolation on the fast algorithms to further decrease errors. Furthermore, we implemented a fast method to determine the roots of the NFT coefficient $a(\lambda)$. These roots are the discrete eigenvalues of the equation and computing them is an essential step to determining the discrete spectrum. We then checked the implementation of the methods by analyzing the errors of the numerical approximations against the number of time domain samples of the potential function. After that, we benchmarked the fast algorithms by providing plots of the error against runtime for multiple test cases and comparing those to the results achieved with slow methods.
In the process of testing the methods we derived the exact solution for the continuous spectrum of the Manakov equation with a rectangle potential. We also determined the exact solution for the continuous spectrum of the Manakov equation with a secant hyperbolic potential.

7-2 Conclusions

Based on the results in Section 6-2 we conclude that the fast NFT algorithms as introduced by Wahls et. al. for the NSE can be and have been succesfully extended to the Manakov equation. For higher numbers of samples and $M = D$, the fast methods give better results than the slow methods as long as the support of the potential function is chosen appropriately and rounding or truncation errors do not start to dominate. We also conclude that fast polynomial rootfinders are an effective method in determining the discrete eigenvalues, although some spurious values are created.

Specifically, for the focusing secant hyperbolic test case without Richardson extrapolation we found the 4split4B, 4split6B and FTES4 with Suzuki factorization methods to give comparable error versus runtime results. With Richardson extrapolation FTES4 with Suzuki factorization slightly outperforms 4split4B and 4split6B for high numbers of time and frequency domain samples. In the defocusing secant hyperbolic test case we see that FTES4 outperforms both the 4split4B and 4split6B methods for the highest numbers of samples, even though it is not significantly better than $\mathrm{CF}_2^{[4]}$. For the rectangle potential test case BO is the best choice as long as the time support T is properly chosen, as the numerical method gives the exact result in that case. From the other tested methods 2split4B and 2split6B are among the best choices for higher numbers of time and frequency domain samples. From the single soliton test case we conclude that FTES4 in not a good choice for computing the discrete eigenvalues. This method is not able to compute the discrete eigenvaues correctly, while all other tested methods do. Unfortunately all methods generate a lot of spurious values as well.

7-3 Recommendations for further additions to the FNFT library

We hope that the FNFT library will be useful in the years to come to anyone working in the field of Nonlinear Fourier Transforms. To this goal, we recommend two additions to the library on the short term. The first is to implement a more computationally efficient multiplication of the transition matrices if D is not a power of 2. A possible method for this was proposed in [10]. We can divide D into groups which are powers of 2:

$$D = 2^{M_1} + 2^{M_2} + \ldots 2^{M_m} \tag{7-1}$$

where each M_i is a positive integer [10]. First fix M_1 as large as possible, then fix M_2 as large as possible and so on. We have thus divided the D samples in m sets which can be multiplied in the tree-like fashion as outlined in Section 4-6, yielding $\mathbf{G}_i(z)$ for $i = 1, 2, ...m$. The final polynomials are then given by $\mathbf{G}(z) = \prod_{i=1}^{m} \mathbf{G}_i(z)$.

The second new feature we recommend implementing is better filtering of the eigenvalues for the Manakov Equation (ME). We noted in the results chapter that a lot of spurious roots of $a(\lambda)$ are identified by the fast rootfinder algorithm. For a higher amount of time samples, these eigenvalues have a large real part, and a possible filtering method would be to simply discard all eigenvalues with a real part above a certain threshold. For lower numbers of time samples this solution will not work.

On the longer term, we recommend extending the methods for computing the discrete spectrum of the NSE to the Manakov equation. A start has been made with the implementation of a fast polynomial rootfinder but more steps need to be taken. The Nonlinear Fourier Transform (NFT) coefficients $b_{1,2}(\lambda)$ and the derivative of NFT coefficient $a(\lambda)$ need to be computed at the values λ_d that were found by the rootfinder. The algorithms for the discrete spectrum of the Manakov equation still need to be developed after which they can be implemented and tested. This falls outside the scope of this thesis project, but we do think this will add greatly to the usefulness of the FNFT library.

Appendix A

MATLAB code for getting the polynomial coefficients

A-1 Code for getting the polynomial coefficients

A-1-1 2split3A coefficients

```
syms q1 q2 l kappa h z...
% split3A method
A = [-1i*l,0,0;0,1i*l,0;0,0,1i*l];
B = [0,q1,q2;-kappa*conj(q1),0,0;-kappa*conj(q2),0,0];
% Let AE = expm(A*h) and BE = expm(B*h)
% Set z = exp(j lambda h/3) such that
AE = [1/z^3,0,0;0,z^3,0;0,0,z^3];
BE = sym('BE',[3,3]);
% CE=expm((A+B)*h)
% Let CE_approx be approximation of CE after application of splitting
% scheme
% AE_3 = expm(A*h/3)
AE_3 = [1/z,0,0;0,z,0;0,0,z];
% BE_3 = expm(B*h/3)
BE_3 = sym('BE_3',[3,3]);

CE_approx = 9/8*AE_3*BE_3*BE_3*AE_3*AE_3*BE_3 - ...
            1/8*AE*BE;

% Dividing throughout by z^-3
CE_approx_pos = expand(CE_approx*z^3);

% We can now look at coefficients of individual polynomials. Here c gives
% the coefficients of the element specified by (i,j) in CE_approx_pos(i,j
    )
% and t gives which power of z the coefficients belong to
```

```
[c11,t11] = coeffs(CE_approx_pos(1,1),z);
[c12,t12] = coeffs(CE_approx_pos(1,2),z);
[c13,t13] = coeffs(CE_approx_pos(1,3),z);

[c21,t21] = coeffs(CE_approx_pos(2,1),z);
[c22,t22] = coeffs(CE_approx_pos(2,2),z);
[c23,t23] = coeffs(CE_approx_pos(2,3),z);

[c31,t31] = coeffs(CE_approx_pos(3,1),z);
[c32,t32] = coeffs(CE_approx_pos(3,2),z);
[c33,t33] = coeffs(CE_approx_pos(3,3),z);

% now look at all the c's to see what the matrices for each coefficient
% should be:
matz0 = [c11(2),    c12(2),    c13(2);
         0,         0,         0;
         0,         0,         0];


matz2 = [0,         0,         0;
         c21(2),    c22(2),    c23(2);
         c31(2),    c32(2),    c33(2)];

matz4 = [c11(1),    c12(1),    c13(1);
         0,         0,         0;
         0,         0,         0];

matz6 = [0,         0,         0;
         c21(1),    c22(1),    c23(1);
         c31(1),    c32(1),    c33(1)];

% now we have the following approximation:
% exp(A+B) = z^-3 * (matz0*1 + matz2*z^2 + matz4*z^4 + matz6*z^6)
```

A-1-2 2split3B coefficients

```
%% computing the coefficients in sybmbolic form
syms q1 q2 l kappa h z

A = [-1i*l,0,0;0,1i*l,0;0,0,1i*l];
B = [0,q1,q2;-kappa*conj(q1),0,0;-kappa*conj(q2),0,0];

% Let AE = expm(A*h) and BE = expm(B*h)
% Set z = exp(j lambda h/3) (m=1/3) such that
AE = [1/z^3,0,0;0,z^3,0;0,0,z^3];
BE = sym('BE',[3,3]);
% AE_3 = expm(A*h/3)
AE_3 = [1/z,0,0;0,z,0;0,0,z];
% BE_1_3 = expm(B*h/3)
BE_1_3 = sym('BE_1_3',[3,3]);
% BE_2_3 = expm(B*h*2/3)
BE_2_3 = sym('BE_2_3',[3,3]);

```

```
18
19 % CE=expm((A+B)*h)
20 % Let CE_approx be approximation of CE after application of splitting
21 % scheme
22
23 CE_approx = (9/8)*BE_1_3*AE_3*AE_3*BE_2_3*AE_3 -...
24             (1/8)*BE*AE;
25 CE_approx_pos = expand(CE_approx*z^3);
26
27 % We can now look at coefficients of individual polynomials. Here c gives
28 % the coefficients of the element specified by (i,j) in CE_approx_pos(i,j
       )
29 % and t gives which power of z the coefficients belong to
30 [c11,t11] = coeffs(CE_approx_pos(1,1),z);
31 [c12,t12] = coeffs(CE_approx_pos(1,2),z);
32 [c13,t13] = coeffs(CE_approx_pos(1,3),z);
33
34 [c21,t21] = coeffs(CE_approx_pos(2,1),z);
35 [c22,t22] = coeffs(CE_approx_pos(2,2),z);
36 [c23,t23] = coeffs(CE_approx_pos(2,3),z);
37
38 [c31,t31] = coeffs(CE_approx_pos(3,1),z);
39 [c32,t32] = coeffs(CE_approx_pos(3,2),z);
40 [c33,t33] = coeffs(CE_approx_pos(3,3),z);
41
42 % Organizing all coefficients in corresponding matrices
43 matz0 = [c11(2),    0,     0;
44          c21(2),    0,     0;
45          c31(2),    0,     0];
46
47
48 matz2 = [0,         c12(2),  c13(2);
49          0,         c22(2),  c23(2);
50          0,         c32(2),  c33(2)];
51
52 matz4 = [c11(1),    0,  0;
53          c21(1),    0,  0;
54          c31(1),    0,  0];
55
56 matz6 = [0,         c12(1),  c13(1);
57          0,         c22(1),  c23(1);
58          0,         c32(1),  c33(1)];
```

A-1-3 2split4A, 4split4A coefficients

```
1 syms q1 q2 l kappa h z...
2 % split4A
3 A = [-1i*l,0,0;0,1i*l,0;0,0,1i*l];
4 B = [0,q1,q2;-kappa*conj(q1),0,0;-kappa*conj(q2),0,0];
5 % Let AE = expm(A*h) and BE = expm(B*h)
6 % Set z = exp(j lambda h/4) such that
7 AE = [1/z^4,0,0;0,z^4,0;0,0,z^4];
8 BE = sym('BE',[3,3]);
```

```
% CE=expm((A+B)*h)
% Let CE_approx be approximation of CE after application of splitting
% scheme
% AE_4 = expm(A*h/4)
AE_4 = [1/z,0,0;0,z,0;0,0,z];
% BE_4 = expm(B*h/4)
BE_2 = sym('BE_2',[3,3]);

CE_approx = (4/3)*AE_4*(BE_2)*(AE_4*AE_4)*(BE_2)*AE_4-...
        (1/3)*(AE_4*AE_4)*BE*(AE_4*AE_4);

% Dividing throughout by z^-4
CE_approx_pos = expand(CE_approx*z^4);

% We can now look at coefficients of individual polynomials. Here c gives
% the coefficients of the element specified by (i,j) in CE_approx_pos(i,j
    )
% and t gives which power of z the coefficients belong to
[c11,t11] = coeffs(CE_approx_pos(1,1),z);
[c12,t12] = coeffs(CE_approx_pos(1,2),z);
[c13,t13] = coeffs(CE_approx_pos(1,3),z);

[c21,t21] = coeffs(CE_approx_pos(2,1),z);
[c22,t22] = coeffs(CE_approx_pos(2,2),z);
[c23,t23] = coeffs(CE_approx_pos(2,3),z);

[c31,t31] = coeffs(CE_approx_pos(3,1),z);
[c32,t32] = coeffs(CE_approx_pos(3,2),z);
[c33,t33] = coeffs(CE_approx_pos(3,3),z);

% now look at all the c's to see what the matrices for each coefficient
% should be:
matz0 = [c11(2) 0       0;...
         0      0       0;...
         0      0       0];

matz2 = [0      c12(3)  c13(3);
         c21(3) 0       0;...
         c31(3) 0       0];

matz4 = [c11(1) c12(2)  c13(2);...
         c21(2) c22(2)  c23(2);...
         c31(2) c32(2)  c33(2)];

matz6 = [0      c12(1)  c13(1);...
         c21(1) 0       0;...
         c31(1) 0       0];

matz8 = [0      0       0;...
         0      c22(1)  c23(1);...
         0      c32(1)  c33(1)];

% now we have the following approximation:
```

```
% exp(A+B) = z^-4 * (matz0*1 + matz2*z^2 + matz4*z^4 + matz6*z^6 + matz8*
    z^8)
```

A-1-4 2split4B, 4split4B coefficients

```
syms s1 s2 l kappa h z

AE = [1/z^2,0,0;0,z^2 0;0,0,z^2];
BE = sym('BE',[3,3]);


% we have CE = expm(A+B)
% Let G_approx be approximation of CE after application of splitting
% scheme, choose z = exp(j*lambda*h/2)
% AE_2 = expm(A*h/2)
% BE_2 = expm(2B*h/4) = expm(B*h/2)
% BE_4 = expm(B*h/4)
AE_2 = [1/z,0,0;0,z,0;0,0,z];
BE_2 = sym('BE_2',[3,3]);
BE_4 = sym('BE_4',[3,3]);

G_approx = ((4/3)*BE_4*AE_2*BE_2*AE_2*BE_4)+...
            -((1/3)*BE_2*AE_2*AE_2*BE_2);

% Dividing throughout by z^-4
G_approx_pos = expand(G_approx*z^2);

% We can now look at coefficients of individual polynomials. Here c gives
% the coefficients of the element specified by (i,j) in CE_approx_pos(i,j
    )
% and t gives which power of z the coefficients belong to
[c11,t11] = coeffs(G_approx_pos(1,1),z);
[c12,t12] = coeffs(G_approx_pos(1,2),z);
[c13,t13] = coeffs(G_approx_pos(1,3),z);

[c21,t21] = coeffs(G_approx_pos(2,1),z);
[c22,t22] = coeffs(G_approx_pos(2,2),z);
[c23,t23] = coeffs(G_approx_pos(2,3),z);

[c31,t31] = coeffs(G_approx_pos(3,1),z);
[c32,t32] = coeffs(G_approx_pos(3,2),z);
[c33,t33] = coeffs(G_approx_pos(3,3),z);
```

A-1-5 2split6B, 4split6B coefficients

```
syms s1 s2 l kappa h z

AE = [1/z^6,0,0;0,z^6,0;0,0,z^6];
BE = sym('BE',[3,3]);


% we have CE = expm(A+B)
```

```
% Let G_approx be approximation of CE after application of splitting
% scheme
% AE_6 = expm(A*h/6)
% then expm(A*h/3) = AE_6*AE_6, expm(A*h/2) = AE_6^3,
% expm(A*h) = AE_12^6
% BE_p = expm(B*h/p)
AE_6 = [1/z,0,0;0,z,0;0,0,z];
BE = sym('BE',[3,3]);
BE_6 = sym('BE_6',[3,3]);
BE_4 = sym('BE_4',[3,3]);
BE_3 = sym('BE_3',[3,3]);
BE_2 = sym('BE_2',[3,3]);

G_approx = (81/40) * BE_6*(AE_6^2*BE_3)^2*AE_6^2*BE_6+...
            -(16/15)*BE_4*AE_6^3*BE_2*AE_6^3*BE_4+...
            (1/24)*BE_2*AE_6^6*BE_2;

% Dividing throughout by z^-6
G_approx_pos = expand(G_approx*z^6);

% We can now look at coefficients of individual polynomials. Here c gives
% the coefficients of the element specified by (i,j) in CE_approx_pos(i,j
    )
% and t gives which power of z the coefficients belong to
[c11,t11] = coeffs(G_approx_pos(1,1),z);
[c12,t12] = coeffs(G_approx_pos(1,2),z);
[c13,t13] = coeffs(G_approx_pos(1,3),z);

[c21,t21] = coeffs(G_approx_pos(2,1),z);
[c22,t22] = coeffs(G_approx_pos(2,2),z);
[c23,t23] = coeffs(G_approx_pos(2,3),z);

[c31,t31] = coeffs(G_approx_pos(3,1),z);
[c32,t32] = coeffs(G_approx_pos(3,2),z);
[c33,t33] = coeffs(G_approx_pos(3,3),z);
```

A-1-6 FTES4_suzuki coefficients

```
%% Matlab file to get the polynomial coefficients for FTES4
% also to generate results for the test file

syms q1 q2 l kappa h z...

A = [-1i*l,0,0;0,1i*l,0;0,0,1i*l];
B = [0,q1,q2;-kappa*conj(q1),0,0;-kappa*conj(q2),0,0];
% Let AE = expm(A*h) and BE = expm(B*h)
% Set z = exp(j lambda h/3) such that
AE = [1/z^3,0,0;0,z^3,0;0,0,z^3];
BE = sym('BE',[3,3]);
% CE=expm((A+B)*h)
% Let CE_approx be approximation of CE after application of splitting
% scheme
% AE_1_3 = expm(A*h/3)
```

```
16 AE_1_3 = [1/z,0,0;0,z,0;0,0,z];
17 AE_m1_3 = [z, 0, 0; 0, 1/z, 0; 0, 0, 1/z];
18 BE7_48_ = sym('BE7_48_',[3,3]);
19 BE3_8_ = sym('BE3_8_',[3,3]);
20 BEm1_48_ = sym('BEm1_48_',[3,3]);
21 E1 = sym('E1',[3,3]);
22 E2 = sym('E2',[3,3]);
23
24 CE_approx = E1*(BE7_48_*AE_1_3*BE3_8_*AE_m1_3*BEm1_48_*...
25     AE*BEm1_48_*AE_m1_3*BE3_8_*AE_1_3*BE7_48_)*E2;
26
27 % Dividing throughout by z^-7 (z^-7 is the lowest power of z found in the
28 % polynomial)
29 CE_approx_pos = expand(CE_approx*z^7);
30
31 % We can now look at coefficients of individual polynomials. Here c gives
32 % the coefficients of the element specified by (i,j) in CE_approx_pos(i,j
       )
33 % and t gives which power of z the coefficients belong to
34 [c11,t11] = coeffs(CE_approx_pos(1,1),z);
35 [c12,t12] = coeffs(CE_approx_pos(1,2),z);
36 [c13,t13] = coeffs(CE_approx_pos(1,3),z);
37
38 [c21,t21] = coeffs(CE_approx_pos(2,1),z);
39 [c22,t22] = coeffs(CE_approx_pos(2,2),z);
40 [c23,t23] = coeffs(CE_approx_pos(2,3),z);
41
42 [c31,t31] = coeffs(CE_approx_pos(3,1),z);
43 [c32,t32] = coeffs(CE_approx_pos(3,2),z);
44 [c33,t33] = coeffs(CE_approx_pos(3,3),z);
45
46 % now look at all the c's to see what the matrices for each coefficient
47 % should be
48 matz0 = [c11(8), c12(8), c13(8);
49          c21(8), c22(8), c23(8);
50          c31(8), c32(8), c33(8)];
51
52 matz2 = [c11(7), c12(7), c13(7);
53          c21(7), c22(7), c23(7);
54          c31(7), c32(7), c33(7)];
55
56 matz4 = [c11(6), c12(6), c13(6);
57          c21(6), c22(6), c23(6);
58          c31(6), c32(6), c33(6)];
59
60 matz6 = [c11(5), c12(5), c13(5);
61          c21(5), c22(5), c23(5);
62          c31(5), c32(5), c33(5)];
63
64 matz8 = [c11(4), c12(4), c13(4);
65          c21(4), c22(4), c23(4);
66          c31(4), c32(4), c33(4)];
67
```

```
68 matz10 = [c11(3), c12(3), c13(3);
69           c21(3), c22(3), c23(3);
70           c31(3), c32(3), c33(3)];
71
72 matz12 = [c11(2), c12(2), c13(2);
73           c21(2), c22(2), c23(2);
74           c31(2), c32(2), c33(2)];
75
76 matz14 = [c11(1), c12(1), c13(1);
77           c21(1), c22(1), c23(1);
78           c31(1), c32(1), c33(1)];
```

Appendix B

Detailed documentation of the library functions

B-0-1 Documentation of the functions

In this section we provide details for all functions used in the Fast Nonlinear Fourier Transform (FNFT) routine for the Manakov equation. Figure B-1 provides a detailed diagram of these functions.

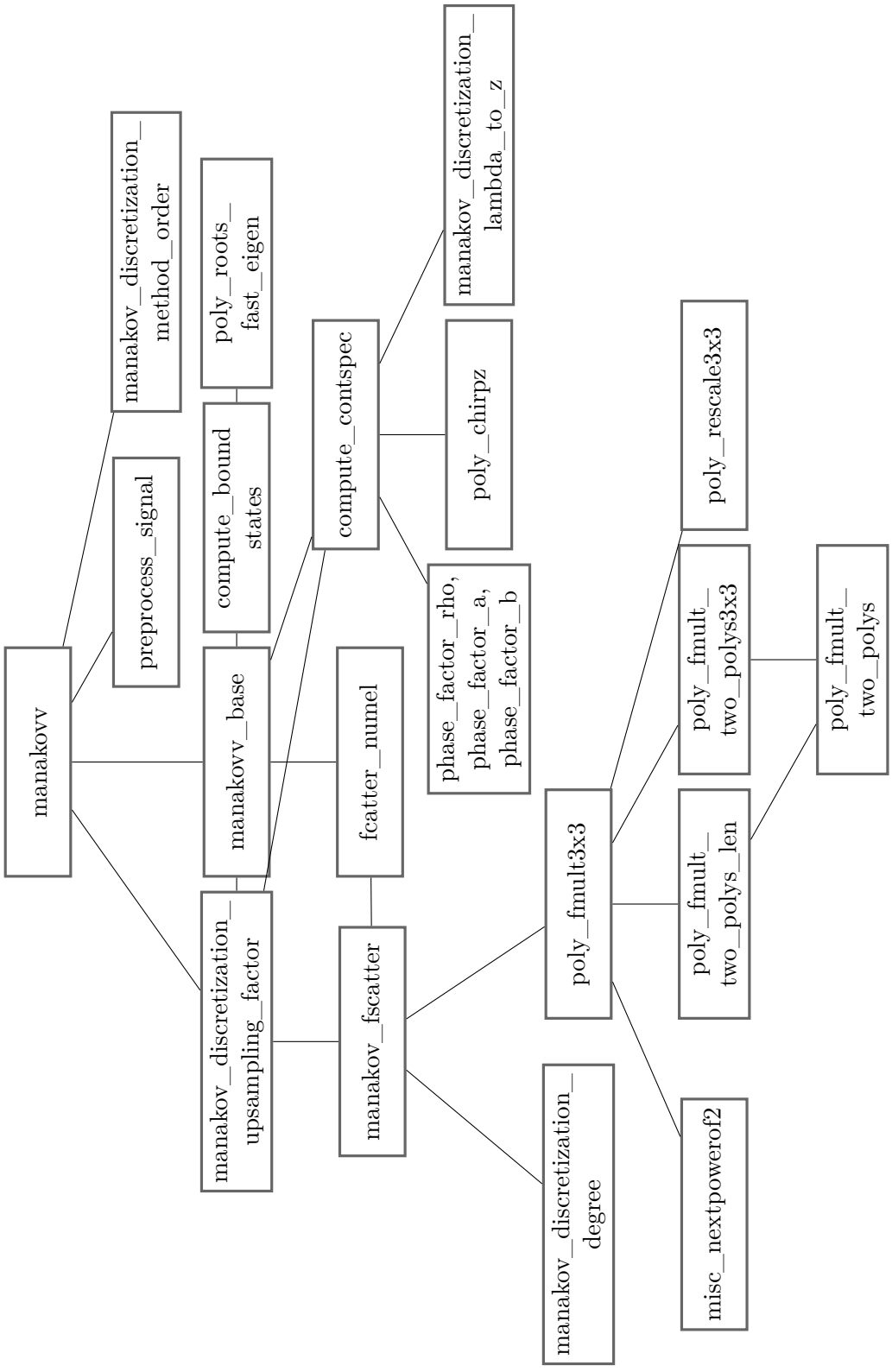


Figure B-1: Detailed diagram of the functions in the library relating to the Manakov equation

manakovv

Inputs

D	Number of time samples from the potential function
T	Array of length 2 indicating the position in time of the first and last potential function sample. C uses 0-relative indexing, so T[0] = T_1 and T[1] = T_2 from Section 3-1
q1, q2	Samples from the potential function: q1 = $q_1(T[0]), q_1(T[0]+h), \ldots q_1(T[1])$ with h being the time step size
M	Desired number of samples in the frequency domain for which the NFT should be determined
XI	Array of length 2 indicating the position of first and last sample in the frequency domain
kappa	Dispersion constant to indicate focussing (kappa = 1) or defocussing (kappa = -1) case
opts	Variable to pass different options to the manakovv routine: discretization method, turn Richardson extrapolation on or off, type of continuous spectrum to be computed (a and $b_{1,2}$ coefficients, $\rho_{1,2}$ coefficients, all coefficients or skip continuous spectrum computation), compute bound states or skip computing bound states

Outputs

contspec	Contains the NFT coefficients a, b_1, b_2 and / or the reflection coefficients 2 depending on the options passed by the user
bound_states	Contains the discrete eigenvalues (roots of $a(\lambda)$)

Highest level function in the library and the only one that is called directly by the user. The user can specify the discretization method, request different NFT coefficients for the continuous spectrum and specify whether the discrete eigenvalues should be calculated using the "opts" input variable.

manakovv_base

Inputs

D	The effective number of samples, that is, the number of transition matrices to be multiplied. This might be different from the original number of samples from the potential function for two possible reasons. If Richardson extrapolation used, manakovv_base is called twice, first using all the available samples and then with only half the samples. For the second call the effective number of samples is halved. See also Section 4-10. Secondly, the fast implementations of $CF_2^{[4]}$ are implemented as if they are fast versions of BO (Section 4-3), but with twice the number of samples. In these cases the effective number of samples is doubled.

q1, q2 The effective samples of the potential function, which are different from the original samples in the second call to manakov_base if Richardson extrapolation is used or if the discretization method is a fast version of $CF_2^{[4]}$. In both cases, preprocess_signal takes care of getting the appropriate samples.

T Array of length 2 indicating the position in time of the first and last sample. Might be different from the original T if Richardson extrapolation is used. Because only half the number of samples is used then, the first sample is still taken at the original T[0], but the last sample used is the sample from T[1]-h

M See inputs manakovv

XI See inputs manakovv

kappa See inputs manakovv

Outputs

contspec See outputs manakovv

bound_states
See outputs manakovv

Performs the actual computations for getting the continuous spectrum and bound states. If a fast method is chosen, manakovv_base calls manakov_fscatter to compute the total transition matrix and passes this matrix to compute_contspec and compute_bound_states. If a slow method is chosen, an empty matrix is passed in place of the transition matrix and multiplications of the transition matrices are done by by compute_contspec directly (bound state localization using a slow method is currently not supported in the library)

discretization_upsampling_factor

Inputs

discretization
Type of discretization used

Output

upsampling_factor
Factor needed to calculate the effective number of samples we should pass to manakovv_base. 2 for (fast discretizations based on) $CF_2^{[4]}$ and 1 for all other discretizations

This function uses a switch-case statement to output a "2" for all (fast discretizations based on) $CF_2^{[4]}$ because these are implemented "as if" they are (fast discretizations based on) BO with double the number of samples. See also Section 4-3 for more details.

preprocess_signal

Inputs

D	Original number of time samples of $\mathbf{q}(x_0, t)$
q1, q2	See inputs manakovv
eps_t	Time step size for the original samples of $\mathbf{q}(x_0, t)$, eps_t= $\frac{T[1]-T[0]}{D-1}$. Called h in the sections of Chapter 3 and Chapter 4 explaining the numerical (F)NFT algorithms
kappa	See inputs manakovv
Dsub	Desired number of samples after subsampling. If Richardson extapolation is used, manakovv_base is called with half the number of samples. In that case we have Dsub = D/2
discretization	see inputs manakovv_base

Outputs

first_last_index: If subsampling is applied, this variable gives the index of the first and last sample from the original potential function samples used

q1_preprocessed, q2_preprocessed
: The samples $\mathbf{q}(x_0, t)$ after preprocessing. They may be subsampled, or different from the actual samples as outlined in Section 4-3 for (fast discretizations based on) $\mathrm{CF}_2^{[4]}$

The function performs some preprocessing on the samples of $\mathbf{q}(x_0, t)$. If subsampling is desired, Dsub < D and preprocess_signal first determines the appropriate number of samples after subsampling based on this desired Dsub. The actual number of subsamples should be such that we can select equidistant samples from the arrays q1, q2. So the actual number of subsamples should be exactly D/n, where n is an integer. Subsampling is required when Richardson extrapolation is used and may also be used when computing the disrete eigenvalues to limit the computation time. After determining the actual Dsub the appropriate samples are selected from q1, q2. first_last_index is set to the first and last index of the original samples that are used for the subsampled samples. If Dsub=D, the subsampling step is skipped.
If the discretization is of type 4splitYZ, 2*Dsub new samples are now determined as outlined in Section 4-3.

manakov_discretization_method_order

Input

discretization
: See inputs manakovv_base

Output

method_order
: Order of the discretization

This function outputs the order of error decay for the chosen discretization method. This is needed for Richardson extrapolation.

manakov_fscatter

Inputs

D	effective number of samples after preprocessing by preprocess_signal
q1, q2	preprocessed samples of the potential function
kappa	See inputs of manakovv
eps_t	time step size for the preprocessed signal samples

discretization
: See inputs discretization_upsampling_factor

Outputs

deg	Polynomial degree of the polynomials in the final transition matrix
result	Array containing the polynomial coefficients of the total transfer matrix, starting with the coefficients of element (1,1) from highest order coefficient to lowest, followed by the coefficients for element (1,2), etc.

If a fast discretization is chosen, manakovv_base calls manakov_fscatter. The function manakov_fscatter first determines and stores the polynomial coefficients for all individual transition. These are a function of the samples, eps_t and the discretization method. They are then multiplied by calling poly_fmult3x3.

discretization_degree

Input

discretization
: See inputs discretization_upsampling_factor

Output

degree	Maximum degree of the polynomials of a single transition matrix for the given discretization

The output degree is used to determine the degree of the total transition matrix. discretization_degree returns 0 if the discretization is a slow method

poly_fmult3x3

Inputs

d	Polynomial degree of a single transition matrix
n	Total number of transition matrices to be multiplied
p	Polynomial coefficients of the transition matrices to be multiplied

Output

result	Resulting polynomial coefficients of the multiplication

This function pads the transition matrices with unity matrices if D is not a power of 2. It then performs tree-wise multiplication by calling poly_fmult_two_polys3x3 for each multiplication in the tree.

poly_fmult_two_polys3x3

Inputs

p1, p2	Polynomial coefficients of the first and second matrix to be multiplied, starting with the polynomial coefficients from element [1,1], then [1,2], ... [3,3]. The first entry corresponds to the coefficient for the highest power of z

Output

result	Resulting polynomial coefficients of the two polynomials that were multiplied

This function performs the 27 polynomial multiplications from Eq. (4-12) needed to multiply two 3x3 matrices. Each of these 27 multiplications is performed by calling poly_fmult_two_polys. To make the code more efficient, poly_fmult_two_polys is called with different modes as argument. If we look at the upper left element of Eq. (4-12), the following steps are carried out:

- call poly_fmult_two_polys to calculate the FFT of $g_{11}h_{11}$
- call poly_fmult_two_polys to calculate the FFT of $g_{12}h_{21}$ and add this to the previously stored FFT of $g_{11}h_{11}$
- call poly_fmult_two_polys to calculate the FFT of $g_{13}h_{31}$, add this to the FFT's of $g_{11}h_{11}$ and $g_{12}h_{21}$, and take the inverse FFT

poly_fmult_two_polys

Inputs

p1, p2	Polynomial coefficients of the two elements to be multiplied, starting with the coefficient for the highest power of z
mode	Indicates what mode is used, see below for explanation of all modes

Output

result	Dependent on mode, this is the result for one element of the matrix after multiplication or an intermediate result

This function takes the FFT of the polynomials passed as inputs and performs the actual multiplication. The output is dependent on the mode.

- Mode 2 takes the FFT of polynomials p1 and p2, multiplies them in the FFT domain and stores this result
- Mode 4 takes the FFT of p1 and p2, multiplies them and adds this to the previously stored value
- Mode 5 takes the FFT of p1 and p2, multiplies them, adds this to the previously stored value and takes the inverse FFT

poly_rescale3x3

Inputs

d	degree of each element of the transition matrix at this stage
p11, p12, ... p33	Arrays with the polynomial coefficients for all elements of the matrix

Output

p11, p12, ...p33	Rescaled arrays
a	Parameter of the scalefactor scl = s^{-a}

poly_fmult3x3 first performs one tree-wise multiplication step i.e. multiplying all pairs of polynomials. If rescaling is desired, poly_fmult3x3 then calls poly_rescale3x3. Function poly_rescale3x3 finds the largest polynomial coefficient of the resulting matrices after multiplication and rescales all coefficients with scl = 2^{-a}, where $a = \log_2(\text{max coeff})$. This ensures the maximum absolute values of the polynomial coefficients stay bounded and prevents overflow errors. Parameter a is then added to the previous value of a. When all multiplications are done, multiplying by a factor s^a gives the original values of the polynomial coefficients.

poly_fmult_two_polys_len

Inputs

deg * n/2	deg is the maximum degree of an element in a single transition matrix, n is the effective amount of matrices to be multiplied

Output

len	length of the buffer

Used to allocate appropriate amount of memory for the buffers used in poly_fmult_two_polys. Output is at least the amount of elements after multiplying two polynomials with length deg*n/2, so at least deg*n. The actual length might be more to allow the third-party code determining the FFT's to work more efficiently.

manakov_compute_contspec

Inputs

deg	Degree of the total transition matrix
W	all the parameters "a" from poly_rescale3x3 added
transfer_matrix	Total transition matrix or NULL if the discretization method is a slow method
q1 q2	Preprocessed samples of the potential. Needed if a slow discretization is chosen
T	See inputs manakovv_base
D	Effective number of samples after preprocessing
XI	See inputs manakovv
M	See inputs manakovv
kappa	See inputs manakovv

Output

result	The requested NFT coefficients, reflection coefficients or both

For fast methods we noted in Section 4-7 that we only need to evaluate elements [1,1], [2,1] and [3,1] of transfer_matrix to get $\mathbf{v}[T2]$. They are evaluated at the desired λ's by first applying the appropriate transformation with manakov_discretization_lambda_to_z and then evaluated using the chirp-Z transform. The NFT coefficients are then given by

$$\begin{aligned} a &= H11 * \text{scl} * \exp(j\lambda * \text{phase\_factor\_a}) \\ b1 &= H21 * \text{scl} * \exp(j\lambda * \text{phase\_factor\_b1}) \\ b2 &= H31 * \text{scl} * \exp(j\lambda * \text{phase\_factor\_b2}) \end{aligned} \tag{B-1}$$

Hij are the elements from the first column of the total transition matrix, scl takes care of possible scaling that was applied, and the term exp($j\lambda$phase_factor) takes care of the multiplication with exp($-j\lambda T1$), multiplication with exp($j\lambda T2$) for the a coefficient and exp($-j\lambda T2$) for the b coefficients, and the extra powers of z we multiplied with in Section 4-2, Section 4-3 and Section 4-4 to make sure we only had positive powers of z. As noted at the end of Section 4-7 these can all be combined in one phase factor.
For a slow method, manakov_scatter_matrix is called which multiplies the numerical values of the scattering matrices for each λ, resulting in the total transition matrix.

manakov_discretization_phase_factor_(rho/a/b)

Inputs

discretization	See inputs manakovv
eps_t	Time step with subsampling taken into account
D	Number of samples after possible subsampling
T	Time instant of first and last sample of the potential function

Output

phase_factor	Phase factor to be used in Eq. (B-1)

This function computes the phase factor.

poly_chirpz

Inputs

deg	Degree of the total transition_matrix
transfer_matrix	Polynomial coefficients of the total transition matrix
A	First λ in the complex plane to evaluate
V	Distance from each point λ to the next
M	Number of points in the frequency domain for which we wish to evaluate the NFT

Output
Hij_vals: values of element ij of polynomial transition matrix evaluated at all desired points

Evaluation of the polynomials for all desired λ's using the chirp-Z transform Section 4-7.

manakov_discretization_lambda_to_z

Inputs

M	Number of points in the frequency domain for which we wish to evaluate the NFT
eps_t	Time step with subsampling taken into account
vals	Upon entry, all λ for which to evaluate the NFT
discretization	See inputs manakovv

Output

vals	Upon exit, all λ transformed to the Z-domain

Determines $z(\lambda)$. For all methods in the library, $z = \exp(j\lambda h/m)$ for a certain integer m. See Section 4-2, Section 4-3 and Section 4-4 for more details on how to choose an appropriate m.

compute_boundstates

Inputs

deg	Polynomial degree of the total transition matrix
transfer_matrix	polynomial coefficients of the total transition matrix
eps_t	Time step with subsampling taken into account

Output

bound_states	The bound states of the NFT for the given potential $\mathbf{q}(x,t)$, i.e., the roots of polynomial $a(\lambda)$

Takes the first element of the total transition matrix and computes the roots of this polynomial using a fast rootfinder. As explained in Section 4-8 these correspond to the roots of polynomial $a(\lambda)$. After that it transforms the found roots z to λ, $\lambda = \frac{m}{jh}\ln(z)$, where m is an integer dependent on the discretization method.

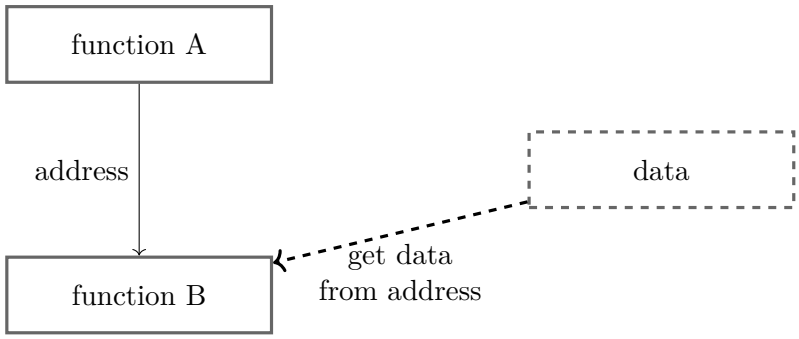


Figure B-2: Using pointers for function input

poly_roots_fast_eigen

inputs

deg Polynomial degree of the total transition matrix

transfer_matrix
Polynomial coefficients of the transfer matrix

output

buffer Array containing the z's for the bound states

This function determines the roots of the upper left element of the transition matrix using the fast polynomial rootfinder found in [5]. These correspond to the roots of $a(\lambda(z))$ and are transformed from z to λ by compute_boundstates.

A note on inputs, outputs and pointers

In the FNFT library, large datastructures are used such as arrays with potential function time samples or the values of the continues spectrum for a high number of samples in the frequency domain. Passing large datastructures from one function to another makes the code slow. In the library this problem is solved by passing a *pointer* to the array instead of the array itself.
If the array is an input, the executing function (B) simply reads the data on the address that A passes to B. See B-2 for a schematic representation. If the array is an output, function A also passes an address to B. This time B writes data to the location indicated by the address, after which A can read out the data on this address. See B-3.
In the description of the functions in Section B-0-1, we did not indicate which pointers are the actual inputs to the functions. Rather, we listed the variables themselves directly as inputs. Secondly, we use "outputs" for the values of interest that are calculated by the function. Most often, the actual output of the function is a return code indicating if any errors occurred and the computed values are stored in a location indicated by a pointer. This choice was made to make the report more readable to some one less familiar with C programming. We direct the reader to the full documentation of the library [1] for full information on this and the datatypes of all variables.

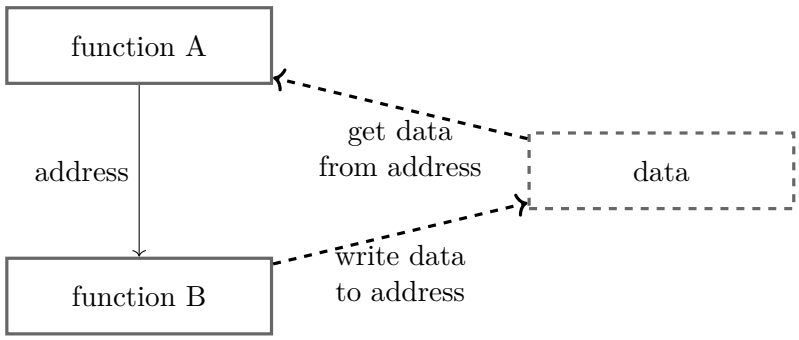


Figure B-3: Using pointers for function output

Bibliography

[1] https://fastnft.github.io/FNFT/. Accessed on 19-07-2021.

[2] FNFT software library. https://github.com/FastNFT/FNFT. Accessed on 27-05-2020.

[3] Erik Agrell, Magnus Karlsson, AR Chraplyvy, David J Richardson, Peter M Krummrich, Peter Winzer, Kim Roberts, Johannes Karl Fischer, Seb J Savory, Benjamin J Eggleton, et al. Roadmap of optical communications. *Journal of Optics*, 18(6), 2016.

[4] Vahid Aref, Henning Bülow, Karsten Schuh, and Wilfried Idler. Experimental demonstration of nonlinear frequency division multiplexed transmission. In *2015 European Conference on Optical Communication (ECOC)*, pages 1–3, 2015.

[5] Jared L. Aurentz, Thomas Mach, Leonardo Robol, Raf Vandebril, and David S. Watkins. Fast and backward stable computation of roots of polynomials, part II: backward error analysis; companion matrix and companion pencil. arXiv preprint: https://arxiv.org/abs/1611.02435, 2018.

[6] Sergio Blanes, Fernando Casas, JA Oteo, and José Ros. The magnus expansion and some of its applications. *Physics reports*, 470(5-6):151–238, 2009.

[7] Guido Boffetta and Alfred Richard Osborne. Computation of the direct scattering transform for the nonlinear schrödinger equation. *Journal of computational physics*, 102(2):252–264, 1992.

[8] S. Burtsev, R. Camassa, and I. Timofeyev. Numerical algorithms for the direct spectral transform with applications to nonlinear schrödinger type systems. *Journal of Computational Physics*, 147(1):166–186, 1998.

[9] Shrinivas Chimmalgi, Peter J. Prins, and Sander Wahls. Nonlinear fourier transform algorithm using a higher order exponential integrator. In *Advanced Photonics 2018 (BGPP, IPR, NP, NOMA, Sensors, Networks, SPPCom, SOF)*. Optical Society of America, 2018. paper SpM4G.5.

[10] Shrinivas Chimmalgi, Peter J. Prins, and Sander Wahls. Fast nonlinear fourier transform algorithms using higher order exponential integrators. *IEEE Access*, 7:145161–145176, 2019.

[11] S. Civelli, S. K. Turitsyn, M. Secondini, and J. E. Prilepsky. Polarization-multiplexed nonlinear inverse synthesis with standard and reduced-complexity nft processing. *Opt. Express*, 26(13):17360–17377, Jun 2018.

[12] Stella Civelli. *Nonlinear frequency-division multiplexing: theoretical aspects, numerical algorithms, and experimental demonstration.* PhD thesis, Sant'Anna School of Advanced Studies, 2019.

[13] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[14] Francesco Da Ros, Stella Civelli, Simone Gaiarin, Edson Porto da Silva, Nicola De Renzis, Marco Secondini, and Darko Zibar. Dual-polarization NFDM transmission with continuous and discrete spectral modulation. *Journal of Lightwave Technology*, 37(10):2335–2343, 2019.

[15] Simone Gaiarin, Auro Michele Perego, Edson Porto da Silva, Francesco Da Ros, and Darko Zibar. Dual-polarization nonlinear fourier transform-based optical communication system. *Optica*, 5(3):263–270, 2018.

[16] Clifford S. Gardner, John M. Greene, Martin D. Kruskal, and Robert M. Miura. Method for solving the korteweg-devries equation. *Phys. Rev. Lett.*, 19:1095–1097, Nov 1967.

[17] Clifford S. Gardner, John M. Greene, Martin D. Kruskal, and Robert M. Miura. Korteweg-devries equation and generalizations. vi. methods for exact solution. *Communications on Pure and Applied Mathematics*, 27(1):97–133, 1974.

[18] Peter D Lax. Integrals of nonlinear equations of evolution and solitary waves. *Communications on pure and applied mathematics*, 21(5):467–490, 1968.

[19] Son Thai Le, Jaroslaw E Prilepsky, and Sergei K Turitsyn. Nonlinear inverse synthesis technique for optical links with lumped amplification. *Optics express*, 23(7):8317–8328, 2015.

[20] Sergei V Manakov. On the theory of two-dimensional stationary self-focusing of electromagnetic waves. *Soviet Physics-JETP*, 38(2):248–253, 1974.

[21] Sergey Medvedev, Igor Chekhovskoy, Irina Vaseva, and Mikhail Fedoruk. Fast computation of the direct scattering transform by fourth order conservative multi-exponential scheme. arXiv preprint: https://arxiv.org/abs/1909.13228, 2019.

[22] Sergey Medvedev, Igor Chekhovskoy, Irina Vaseva, and Mikhail Fedoruk. Conservative multi-exponential scheme for solving the direct zakharov–shabat scattering problem. *Opt. Lett.*, 45(7):2082–2085, Apr 2020.

[23] Sergey Medvedev, Irina Vaseva, Igor Chekhovskoy, and Mikhail Fedoruk. Novel Numerical algorithm with fourth-order accuracy for the direct Zakharov-Shabat problem. *Optics Letters*, 44(9):2264, May 2019.

[24] Sergey Medvedev, Irina Vaseva, Igor Chekhovskoy, and Mikhail Fedoruk. Exponential fourth order schemes for direct zakharov-shabat problem. *Optics Express*, 28(1):20–39, 2020.

[25] CR Menyuk. Application of multiple-length-scale methods to the study of optical fiber transmission. *Journal of Engineering Mathematics*, 36(1):113–136, 1999.

[26] Borislav V Minchev and Will Wright. A review of exponential integrators for first order semi-linear problems. Technical report, Norwegion University of Science and Technology, 2005.

[27] Jaroslaw Prilepsky, Stanislav Derevyanko, Keith Blow, Ildar Gabitov, and S.k Turitsyn. Nonlinear inverse synthesis and eigenvalue division multiplexing in optical fiber channels. *Physical Review Letters*, 113:013901, 07 2014.

[28] Peter J. Prins and Sander Wahls. Higher order exponential splittings for the fast nonlinear fourier transform of the korteweg-de vries equation. In *Proceedings 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4524–4528, United States, 2018. IEEE. Accepted Author Manuscript.

[29] L. Rabiner, R. Schafer, and C. Rader. The chirp z-transform algorithm. *IEEE Transactions on Audio and Electroacoustics*, 17(2):86–92, 1969.

[30] Junkichi Satsuma and Nobuo Yajima. B. initial value problems of one-dimensional self-modulation of nonlinear waves in dispersive media. *Progress of Theoretical Physics Supplement*, 55:284–306, 1974.

[31] C. E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[32] Masuo Suzuki. General nonsymmetric higher-order decomposition of exponential operators and symplectic integrators. *Journal of the Physical Society of Japan*, 61(9):3015–3019, 1992.

[33] Mechthild Thalhammer. A fourth-order commutator-free exponential integrator for nonautonomous differential equations. *SIAM Journal on Numerical Analysis*, 44:851–864, 2006.

[34] Takayuki Tsuchida. N-Soliton Collision in the Manakov Model. *Progress of Theoretical Physics*, 111(2):151–182, 02 2004.

[35] Sergei K Turitsyn, Jaroslaw E Prilepsky, Son Thai Le, Sander Wahls, Leonid L Frumin, Morteza Kamalian, and Stanislav A Derevyanko. Nonlinear fourier transform for optical data processing and transmission: advances and perspectives. *Optica*, 4(3):307–322, 2017.

[36] A. Vasylchenkova, J.E. Prilepsky, D. Shepelsky, and A. Chattopadhyay. Direct nonlinear fourier transform algorithms for the computation of solitonic spectra in focusing nonlinear schrödinger equation. *Communications in Nonlinear Science and Numerical Simulation*, 68:347 – 371, 2019.

[37] C. Vuik, F.J. Vermolen, M.B. Gijzen, and M.J. Vuik. *Numerical Methods for Ordinary Differential Equations*. Delft Academic Press, 2015.

[38] S. Wahls and H. V. Poor. Introducing the fast nonlinear fourier transform. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5780–5784, May 2013.

[39] Sander Wahls, Shrinivas Chimmalgi, and Peter Prins. Fnft: A software library for computing nonlinear fourier transforms. *Journal of Open Source Software*, 3(23):597, 2018.

[40] Sander Wahls and H Vincent Poor. Fast numerical nonlinear fourier transforms. *IEEE Transactions on Information Theory*, 61(12):6957–6974, 2015.

[41] Eric W. Weisstein. Runge-kutta method. https://mathworld.wolfram.com/Runge-KuttaMethod.html. Accessed on 27-04-2020.

[42] T. Wolinski. Polarization in optical fibers. *Acta Physica Polonica A*, 95:749–760, 1999.

[43] P. Wormer. Electromagnetic wave. https://commons.wikimedia.org/w/index.php?curid=12270410. This file is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license. Accessed on 01-12-2020.

[44] M. I. Yousefi and F. R. Kschischang. Information transmission using the nonlinear fourier transform, part i: Mathematical tools. *IEEE Transactions on Information Theory*, 60(7):4312–4328, July 2014.

[45] M. I. Yousefi and F. R. Kschischang. Information transmission using the nonlinear fourier transform, part ii: Numerical methods. *IEEE Transactions on Information Theory*, 60(7):4329–4345, July 2014.

[46] V. E. Zaharov and A. B. Shabat. Exact theory of two-dimensional selffocusing and one-dimensional selfmodulation of waves in nonlinear media. *Sov. Phys. JETP*, 34:62–69, 1972.

Glossary

List of Acronyms

ADC	Analog-to-Digital Converter
DCSC	Delft Center for Systems and Control
BO	Boffetta-Osborne
DAC	Digital-to-Analog converter
DE	Differential Equation
FFT	Fast Fourier Transform
FNFT	Fast Nonlinear Fourier Transform
FT	Fourier Transform
IST	Inverse Scattering Transform
KdV	Korteweg-de Vries
MZS	Manakov Zakharov Shabat
NSE	Nonlinear Schrödinger Equation
NFT	Nonlinear Fourier Transform
NFDM	Nonlinear Frequency Division Multiplexing
ME	Manakov Equation
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
RE	Richardson Extrapolation
RK4	fourth-order Runge-Kutta
ZS	Zakharov Shabat