Fast Nonlinear Fourier Transform Algorithms Using Higher Order Exponential Integrators

SHRINIVAS CHIMMALGI [ID], **PETER J. PRINS** [ID], **AND SANDER WAHLS** [ID], **(Senior Member, IEEE)**
Delft Center for Systems and Control, Delft University of Technology, 2628 CD Delft, The Netherlands
Corresponding author: Shrinivas Chimmalgi (s.chimmalgi@tudelft.nl)

ABSTRACT The nonlinear Fourier transform (NFT) has recently gained significant attention in fiber optic communications and other engineering fields. Although several numerical algorithms for computing the NFT have been published, the design of highly accurate low-complexity algorithms remains a challenge. In this paper, we present new fast forward NFT algorithms that achieve accuracies that are orders of magnitudes better than current methods, at comparable run times and even for moderate sampling intervals. The new algorithms are compared to existing solutions in multiple, extensive numerical examples.

INDEX TERMS Nonlinear Fourier transform, transforms for signal processing, fast algorithms for DSP, nonlinear signal processing.

I. INTRODUCTION

The fast Fourier transform (FFT) is a well-known success story in engineering. From a numerical point of view, the FFT provides a mere first-order approximation of the discrete-time Fourier transform one is actually interested in. Hence the success of the FFT is quite surprising. Upon closer inspection, it however turns out that approximations based on FFTs are very accurate if the signal is smooth [1]. Recently, *nonlinear Fourier transforms* (NFTs) have been gaining much attention in engineering areas such as fiber-optic communications [2], [3] and coastal engineering [4], [5]. NFTs are generalizations of the conventional Fourier transform that allow to solve specific nonlinear evolution equations in a way that is analogous to how Fourier solved the heat equation [6]. The evolution of the nonlinear Fourier spectrum is, exactly like in the linear case, much simpler than the evolution of the original signal. NFTs also have unique data analysis capabilities that enable the detection of particle-like signal components known as solitons [7].

Recently, a nonlinear variant of the FFT has been derived [8], [9]. These type of fast NFTs (FNFTs) can provide up to second-order accuracy [3]. Unfortunately, unlike for the FFT, the accuracy of the FNFTs in [3], [8], [9] remains (at most) second-order even when the signal is smooth.

The associate editor coordinating the review of this manuscript and approving it for publication was Jinming Wen [ID].

As a result, engineers currently have to strongly oversample even smooth signals in order to get reliable numerical results [10, Section 4]. Several authors have proposed NFT algorithms with higher orders of accuracy, utilizing either Runge-Kutta [11], [12] or implicit Adams methods [13]. However, even though these methods have higher accuracy orders, they require very small sampling intervals in order to actually perform better than standard second-order method such as [14]. For practically relevant sampling intervals, they are typically not the best choice as they are slower *and* may even perform worse in these regimes. Numerical methods that provide better complexity-accuracy trade-offs in practically relevant regimes have been an open problem until recently.

In [15], the authors introduced a new numerical method that can compute the NFT with accuracies that are orders of magnitudes better than those of standard methods while having comparable run times. The key enabler for this large improvement in the complexity-accuracy trade-off was that, for the first time, a so-called commutator-free exponential integrator [16] of higher order was used to compute the NFT. In a nutshell, the absence of commutator terms drastically reduces the computational cost whereas the excellent performance of exponential integrators is retained. However there is one drawback remaining in [15]: The complexity of the algorithm grows quadratically with the number of signal samples D, which makes the algorithm attractive only if the number of samples is not too high. In other words the algorithm is

not fast. In this paper we overcome this limitation. **Our main contribution is the first fast higher-order NFT algorithm based on an exponential integrator. By combining it with Richardson extrapolation scheme, we arrive at an NFT algorithm that requires only $\mathcal{O}(D \log^2 D)$ floating point operations, but achieves a sixth-order $[\mathcal{O}(D^{-6})]$ error decay.[1] To the best of our knowledge no such algorithm has been investigated in the literature before. We show that the complexity-accuracy trade-off of the proposed algorithm is dramatically better than that of existing standard methods.** To give an illustration, we point out that in one of our numerical examples, our new method achieves an accuracy that is *hundred million* times better than the standard second-order method in [14] at a comparable run time.[2]

The rest of this paper is structured as follows. In Section II we recapitulate the required mathematical background of the NFT. In Section III we derive improved versions of our recently proposed numerical NFT in [15], and compare them with both conventional second-order and other higher-order NFT algorithms in multiple numerical examples. Then, in Section IV, we demonstrate how some of our new numerical NFTs can be made fast. The fast versions are compared to their slow counterparts. Next, in Section V, we investigate how Richardson extrapolation can improve the complexity-accuracy trade-off of the fast NFT methods even further. The paper is finally concluded in Section VI.[3]

Notation: Real numbers: $\mathbb{R}$; $\mathbb{R}_{\geq 0} := \{x \in \mathbb{R} : x \geq 0\}$; Complex numbers: $\mathbb{C}$; Complex numbers with positive imaginary part: $\mathbb{H}$; Integers: $\mathbb{Z}$; $i := \sqrt{-1}$; Euler's number: e; Real part: $\mathrm{Re}(\cdot)$; Complex conjugate: $(\cdot)^*$; Floor function: $\lfloor \cdot \rfloor$; Absolute value: $|\cdot|$; Matrix exponential: $\mathrm{expm}(\cdot)$; Matrix product: $\prod_{k=1}^{K} A_k := A_K A_{K-1} \times \cdots \times A_1$; Matrix element in the ith column and jth row: $[\cdot]_{i,j}$; Fourier transform of the function $f(t)$, $\mathcal{F}(f(t)) = \tilde{f}(\xi) = \int_{-\infty}^{\infty} f(t) e^{-it\xi} \, dt$; Inverse Fourier transform of the function $\tilde{f}(\xi)$, $\mathcal{F}^{-1}(\tilde{f}(\xi)) = f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \tilde{f}(\xi) e^{it\xi} \, d\xi$.

II. PRELIMINARIES

In this section we describe the mathematical machinery behind the nonlinear Fourier transform (NFT). For illustration purposes we will describe the NFT in the context of fiber-optic communications. Let $q(x, t)$ denote the complex envelope of the electric field in an ideal optical fiber, whose evolution in normalized coordinates is described by the nonlinear Schrödinger equation (NSE) [17, Chap. 2]

$$i\frac{\partial q}{\partial x} + \frac{\partial^2 q}{\partial t^2} + 2\kappa |q|^2 q = 0. \quad (1)$$

---

[1] The complexity estimate only contains the cost of computing the so-called continuous spectrum as is usual in the NFT literature. Details on the continuous spectrum will be given later in the text. The cost of computing the discrete spectrum are highly problem dependent.

[2] Compare the error for $\mathrm{CF}_1^{[2]}$ in Fig. 6 with that of $\mathrm{FCF\_RE}_2^{[4]}$ in Fig. 13 for the execution time 1 sec. We remark that although the execution times are implementation specific, they still give a good indication of the advantages of our proposed algorithm (see Appendix A).

[3] Some of the results were presented at the OSA Advanced Photonics Congress, Zurich, July 2018 (SpM4G.5).

Here, $x \geq 0$ denotes the location in the fiber and t denotes retarded time. The parameter κ determines if the dispersion in the fiber is normal (-1) or anomalous $(+1)$. When $\kappa = +1$, (1) is referred to as the focusing NSE and for $\kappa = -1$ (1) is referred to as the defocusing NSE. The NFT that solves the NSE (1) is due to Zakharov and Shabat [18]. It transforms any signal $q(t)$ that vanishes sufficiently fast for $t \to \pm\infty$ from the time-domain to the nonlinear Fourier domain through the analysis of the linear ordinary differential equation (ODE)

$$\frac{\partial V(t, \lambda)}{\partial t} = C(t, \lambda) V(t, \lambda) = \begin{bmatrix} -i\lambda & q(t) \\ -\kappa q^*(t) & i\lambda \end{bmatrix} V(t, \lambda), \quad (2)$$

where $q(t) = q(x_0, t)$ for any fixed x_0, subject to the boundary conditions

$$V(t, \lambda) = [\phi(t, \lambda) \, \bar{\phi}(t, \lambda)]$$
$$\to \begin{bmatrix} e^{-i\lambda t} & 0 \\ 0 & -e^{i\lambda t} \end{bmatrix} \text{ as } t \to -\infty,$$
$$V(t, \lambda) = [\bar{\psi}(t, \lambda) \, \psi(t, \lambda)]$$
$$\to \begin{bmatrix} e^{-i\lambda t} & 0 \\ 0 & e^{i\lambda t} \end{bmatrix} \text{ as } t \to \infty. \quad (3)$$

The term $\lambda \in \mathbb{C}$ is a spectral parameter similar to s in the Laplace domain. The matrix $V(t, \lambda)$ is said to contain the eigenfunctions since (2) can be rearranged into an eigenvalue equation with respect to λ [6]. One can view the eigenfunctions $V(t, \lambda)$ as being scattered by $q(t)$ as they move from $t \to -\infty$ to $t \to \infty$. Hence (2) is referred to as the scattering problem [18]. (Many problems in signal processing can be expressed through such scattering problems [19].) For (2) subject to boundary conditions (3), there exists a unique matrix

$$S(\lambda) = \begin{bmatrix} a(\lambda) & \bar{b}(\lambda) \\ b(\lambda) & -\bar{a}(\lambda) \end{bmatrix}, \quad (4)$$

called the scattering matrix, such that [6]

$$[\phi(t, \lambda) \, \bar{\phi}(t, \lambda)] = [\bar{\psi}(t, \lambda) \, \psi(t, \lambda)] S(\lambda). \quad (5)$$

The components $a(\lambda)$, $b(\lambda)$, $\bar{b}(\lambda)$ and $\bar{a}(\lambda)$ are known as the scattering data. The components $a(\lambda)$ and $b(\lambda)$ are sufficient to describe the signal completely. Their evolution along the x dimension (along the length of the fiber) is simple [6, Section III]

$$a(x, \lambda) = a(0, \lambda),$$
$$b(x, \lambda) = b(0, \lambda) e^{-4i\lambda^2 x}. \quad (6)$$

The *reflection coefficient* is then defined as $\rho(\lambda) = b(\lambda)/a(\lambda)$ for $\lambda \in \mathbb{R}$ and it represents the continuous spectrum. In the case of $\kappa = 1$, the nonlinear Fourier spectrum can also contain a so-called *discrete spectrum*. It corresponds to the complex poles of the reflection coefficient in the upper half-plane $\mathbb{H}$, or equivalently to the zeros $\lambda_k \in \mathbb{H}$ of $a(\lambda)$. It is known that there are only finitely many (N) such poles. The poles λ_k are also referred to as eigenvalues and a corresponding set of values $\rho_k := b(\lambda_k) / \left. \frac{da(\lambda)}{d\lambda} \right|_{\lambda = \lambda_k}$ are known

as residues [6, App. 5]. There are different ways to define a nonlinear Fourier spectrum. One possibility is $\{\rho(\lambda)\}_{\lambda \in \mathbb{R}}$, $(\lambda_k, \rho_k)_{k=1}^N$ [6]. The other is $\{b(\lambda)\}_{\lambda \in \mathbb{R}}$, $(\lambda_k, b(\lambda_k))_{k=1}^N$ [20]. In this paper we are primarily interested in computation of $\rho(\lambda)$ but some notes regarding computation of $b(\lambda)$ and the λ_k will also be given. Although we will illustrate our algorithms by applying them to the specific case of NFT of NSE with vanishing boundary condition, it should be noted that we in fact presenting algorithms for solving a class of equations similar to (2) [6, Eq. 2]. Hence the algorithms presented in this paper should carry over to NFTs w.r.t. other nonlinear evolution equations such as the Korteweg–de Vries equation [21] and other boundary conditions.

III. NUMERICAL COMPUTATION OF NFT USING HIGHER ORDER EXPONENTIAL INTEGRATORS

In this section we will start by outlining some assumptions that are required for the numerical methods that will be presented. We will give a brief overview of one of the approaches for computing the NFT and then talk specifically about implementations using commutator-free exponential integrators. To evaluate the methods, we describe examples and performance criteria. We will finally show and compare the results for various methods applied to the mentioned examples.

We remark that only one of the investigated commutator-free exponential integrators can later serve as basis for our new fast method. However, the remaining higher order integrators have their own merits when the number of samples is low, since (asymptotically) slow NFT algorithms can be faster than (asymptotically) fast NFT algorithms in that regime.

A. ASSUMPTIONS
Just like the FFT, the numerical computation of the NFT is carried out with finitely many discrete data samples. Hence, we need to make the following assumptions:

1) The support of the signal $q(t)$ is truncated to a finite interval, $t \in [T_-, T_+]$, instead of $t \in (-\infty, \infty)$. The values $T_\pm$ are chosen such that the resulting truncation error is sufficiently small. The approximation is exact if $q(t) = 0 \ \forall \ t \notin [T_-, T_+]$.

2) The interval $[T_-, T_+]$ is divided into D subintervals of width $h = (T_+ - T_-)/D$. We assume that the signal is sampled at the midpoints of each subinterval $t_n = T_- + (n + 0.5)h$, $n = 0, 1, \ldots, D - 1$ such that $q_n := q(t_n)$.

B. NUMERICAL SCATTERING
The main step in numerically computing the NFT is to solve the scattering problem (2) for $\phi(T_+, \lambda)$ for different values of λ. We can view the D subintervals as layers which scatter the eigenfunction $\phi(t, \lambda)$ as it moves from $t = T_-$ to $t = T_+$. Using numerical ODE solvers we solve for an approximation $\hat{\phi}(T_+, \lambda)$ of $\phi(T_+, \lambda)$. By taking $\bar{\psi}(T_+, \lambda)$ and $\psi(T_+, \lambda)$ equal to the limit in (3) at $t = T_+$, we can compute with (5)

a numerical approximation of the scattering data and ultimately the reflection coefficient.

C. EXPONENTIAL INTEGRATORS
Almost any numerical method available in literature for solving ODEs can be used to solve for $\phi(T_+, \lambda)$ [11], [22]. However, we are particularly interested in so-called exponential type integrators. These methods have been shown to provide a very good trade-off between accuracy and computational cost in several numerical benchmark problems while being fairly easy to implement, see [23] and references therein. We propose to use a special sub-class known as commutator-free quasi-Magnus (CFQM) exponential integrators as some NFT algorithms based on these integrators turn out to have the special structure [8] that is needed to make them fast. We show this in Section IV.

The results in [24] provide a scheme to compute a numerical approximation $\hat{\phi}(T_+, \lambda)$ of $\phi(T_+, \lambda)$. We start by fixing $\hat{\phi}(T_+, \lambda) = H(\lambda)\phi(T_-, \lambda)$, where

$$H(\lambda) = \left(\prod_{n=0}^{D-1} G_n(\lambda) \right)$$
$$= G_{D-1}(\lambda) \, G_{D-2}(\lambda) \cdots G_0(\lambda), \qquad (7)$$

with n being the index of samples of $q(t)$.

The structure of $G_n(\lambda)$ depends on the integrator and the exact values depend on the signal samples q_n and the value of λ. For the integrator in [24], $G_n(\lambda) = \mathrm{CF}_J^{[r]}(t_n, \lambda)$ which leads to the following iterative scheme:

$$\hat{\phi}_{n+1}(\lambda) = \mathrm{CF}_J^{[r]}(t_n, \lambda)\phi_n(\lambda)$$
$$= \prod_{j=1}^{J} \mathrm{expm}(B_j(t_n, \lambda))\phi(t_n, \lambda)$$
$$= \mathrm{expm}(B_J(t_n, \lambda)) \cdots \mathrm{expm}(B_1(t_n, \lambda))\phi_n(\lambda)$$
$$= \phi(t_{n+1}, \lambda) + \mathcal{O}(h^{r+1}), \qquad (8)$$

where expm is the matrix exponential

$$B_j(t_n, \lambda) = h \sum_{k=1}^{K} a_{jk} C_k(t_n, \lambda), \quad j \in \{1, \ldots, J\},$$
$$C_k(t_n, \lambda) = C(t_n + (c_k - 0.5)h, \lambda), \qquad (9)$$

where a_{jk} and $c_k \in [0, 1]$ for $k \in \{1, \ldots, K\}$ are constants that are specific to the integrator and $C(t, \lambda)$ as in (2). By iterating with (7) from $n = 0, 1, \ldots, D - 1$, we obtain the numerical approximation of $\phi(T_+, \lambda)$ that we need to compute the NFT.

For an integrator $\mathrm{CF}_J^{[r]}$, r is the order and J is the number of matrix exponentials required for each subinterval. K is the number of points within each subinterval where the signal value needs to be known. We refer the reader to [24] for their derivation.

An integrator of order r has a local error (error in each subinterval) of $\mathcal{O}(h^{r+1})$. Over $D \ (\propto 1/h)$ such subintervals i.e., over the interval $[T_-, T_+]$, the global error will be $\mathcal{O}(h^r)$.

This distinction of local and global error will become important when we define the error metric used to compare the various algorithms in Section III-D.

The integrator $CF_1^{[2]}$ is also sometimes referred to as the exponential midpoint rule. It was used in the context of NFT for the defocusing NSE ($\kappa = -1$) by Yamada and Sakuda [25] and later by Boffetta and Osborne [14]. For $CF_1^{[2]}$, (8) reduces to

$$\hat{\phi}_{n+1}(\lambda) = G_n(\lambda)\phi_n(\lambda), \quad \text{where}$$
$$G_n(\lambda) = \text{expm}(hC_n(\lambda)),$$
$$C_n(\lambda) = \begin{bmatrix} -i\lambda & q_n \\ -\kappa q_n^* & i\lambda \end{bmatrix}. \quad (10)$$

This is applied repeatedly as in (7) to obtain $\hat{\phi}(T_+, \lambda)$. In [15] we investigated the possibility of using $CF_2^{[4]}$ (first introduced in [16]) to obtain $\hat{\phi}(T_+, \lambda)$. We were able to show its advantage over $CF_1^{[2]}$ when considering the trade-off between an error and execution time. Here we investigate further in this direction and evaluate $CF_3^{[4]}$, $CF_3^{[5]}$ and $CF_4^{[6]}$, which are fourth-, fifth- and sixth-order methods respectively.

The CFQM exponential integrators require multiple non-equispaced points within each subinterval. However, it is unrealistic to assume that signal samples at such non-equispaced points can be obtained in a practical setting. In [15] we used local cubic-spline based interpolation to obtain the non-equispaced points from the mid-points of each subinterval. (We will refer to the samples at these mid-points as the given samples.) However we found that local cubic-spline based interpolation is not accurate enough for higher-order methods. Here, we propose to utilize the Fourier transform and its time-shift property for interpolation, i.e.,

$$q(t - t_s) = \mathcal{F}^{-1}(e^{-i\xi t_s}\mathcal{F}(q(t))), \quad (11)$$

to obtain the samples on shifted time grids required for (9) with $t_s = -(c_k - 0.5)h$. This interpolation rule is also known in signal-processing literature as sinc or bandlimited interpolation [26, Section 7.4.2] and it is accurate when $q(t)$ is sampled in accordance with the Nyquist criterion. As we are working with discrete signal samples, the interpolation can be implemented efficiently using the FFT. The MATLAB code that we used can be found in Appendix B. We remark that we use band-limited interpolation for all methods that require non-equispaced samples: $CF_2^{[4]}$, $CF_3^{[4]}$, $CF_3^{[5]}$ and $CF_4^{[6]}$.

D. ERROR METRIC AND NUMERICAL EXAMPLES
In this subsection, we compare the performance of CFQM exponential integrators $CF_1^{[2]}$, $CF_2^{[4]}$, $CF_3^{[4]}$, $CF_3^{[5]}$ and $CF_4^{[6]}$, the two-step Implicit-Adams method (IA$_2$) introduced in [13] and the fourth-order Runge-Kutta method [11] (RK$_4$) for computation of the reflection coefficient. The fourth-order Runge-Kutta method ($r = 4$) was the first fourth-order method used for the computation of the reflection coefficient in [11], [12]. We include the third-order two-step Implicit-Adams method ($r = 3$) here as it was the first higher-order method that was introduced in the context of fast nonlinear

Fourier transform. The meaning of "fast" will be made precise in Section IV.

We are interested in evaluating the trade-off between the increased accuracy and execution time due to use of higher-order methods. We assess the accuracy of different methods using the relative L^2-error

$$E_\rho = \frac{\sqrt{\sum_{n=0}^{M-1} |\rho(\lambda_n) - \hat{\rho}(\lambda_n)|^2}}{\sqrt{\sum_{n=0}^{M-1} |\rho(\lambda_n)|^2}}, \quad (12)$$

where $\rho(\lambda)$ is the analytical reflection coefficient, $\hat{\rho}(\lambda)$ is the numerically computed reflection coefficient and λ_n are M equally-spaced points in $[-\lambda_{\max}, \lambda_{\max}]$. E_ρ is a global error and hence it is expected to be $\mathcal{O}(h^r)$ for an integrator of order r as explained in Section III-C. We compute the reflection coefficient at the same number of points M as the number of given samples D, i.e. $M = D$, unless mentioned explicitly otherwise.

1) EXAMPLE 1: HYPERBOLIC SECANT, $\kappa = 1$
As the first numerical example we chose the signal $q(t) = \mathring{q}e^{-2i\lambda_0 t}\text{sech}(t)$. The closed form of the reflection coefficient is given by applying the frequency-shift property [27, Section D] to the analytical known reflection coefficient of the secant-hyperbolic signal [28],

$$\rho(\lambda) = \frac{b(\lambda)}{a(\lambda)},$$
$$b(\lambda) = \frac{-\sin(\mathring{q}\pi)}{\cosh(\pi(\lambda - \lambda_0))},$$
$$a(\lambda) = \frac{\Gamma^2(0.5 - i\lambda)}{\Gamma(0.5 - i(\lambda - \lambda_0) + \mathring{q})\Gamma(0.5 - i(\lambda - \lambda_0) - \mathring{q})}, \quad (13)$$

where $\Gamma(\cdot)$ is the gamma function. The discrete spectrum is

$$\lambda_k = \lambda_0 + i(\mathring{q} + 0.5 - k), \quad k = 1, 2, \ldots, M_D, \quad (14)$$
$$b_k = (-1)^k, \quad k = 1, 2, \ldots, M_D, \quad (15)$$
$$M_D = \lfloor(\mathring{q} + 0.5)\rfloor. \quad (16)$$

We set $\mathring{q} = 5.4$, $\lambda_0 = 3$, $\lambda_{\max} = 10$, and chose $[T_-, T_+] = [-32, 32]$ to ensure negligible truncation error.

2) EXAMPLE 2: RATIONAL REFLECTION COEFFICIENT WITH ONE POLE, $\kappa = 1$
The signal is given by [29]

$$q(t) = \begin{cases} -2i\gamma\dfrac{\alpha}{|\alpha|}\text{sech}\left(2\gamma t + \text{arctanh}\left(\dfrac{\beta}{\gamma}\right)\right), & t \leq 0 \\ 0, & t > 0, \end{cases} \quad (17)$$

where α, β are scalar parameters and $\gamma = \sqrt{\alpha\alpha^* + \beta^2}$. We used $\alpha = 1$ and $\beta = -1$. The corresponding reflection coefficient is then known to be

$$\rho(\lambda) = \frac{\alpha}{\lambda - i\beta}. \quad (18)$$

We set $\lambda_{\max} = 60$ and chose $[T_-, T_+] = [-12, 0]$. As the signal in (17) has a discontinuity, it cannot be interpolated well using bandlimited interpolation. We hence assume only in this example that we can sample the signal at exactly the points that we require.

3) EXAMPLE 3: HYPERBOLIC SECANT, $\kappa = -1$

The signal is given by

$$q(t) = \frac{Q}{L}\left(\mathrm{sech}\left(\frac{t}{L}\right)\right)^{1-2i\mathcal{G}}, \quad (19)$$

where $\mathcal{G}$, $\mathcal{L}$ and $\mathcal{Q}$ are scalar parameters. We used $\mathcal{G} = 1.5$, $\mathcal{L} = 0.04$ and $\mathcal{Q} = 5.5$. The corresponding reflection coefficient is known to be [30]

$$\rho(\lambda) = -2^{-2i\mathcal{G}}\mathcal{Q}\frac{\Gamma(d)\Gamma(f_-)\Gamma(f_+)}{\Gamma(d^*)\Gamma(g_-)\Gamma(g_+)}, \quad (20)$$

where $\Gamma(\cdot)$ is the gamma function, $d = 0.5 + i(\lambda\mathcal{L} - \mathcal{G})$, $f_\pm = 0.5 - i(\lambda\mathcal{L} \pm \sqrt{\mathcal{G}^2 + \mathcal{Q}^2})$, and $g_\pm = 1 - i(\mathcal{G} \pm \sqrt{\mathcal{G}^2 + \mathcal{Q}^2})$. We set $\lambda_{\max} = 250$ and chose $[T_-, T_+] = [-1.5, 1.5]$.

The numerical methods were implemented and tested in 64-bit MATLAB (R2018a) running in Ubuntu 16.04 on a machine with an Intel® Core™ i7-5600U CPU with a maximum clock rate of 3200 MHz and 8192 MB of DDR3 memory at 1600 MHz. The CPU was set to the highest available performance setting and the number of computational threads was set to 1 using the `maxNumCompThreads` function of MATLAB. The closed-form expression of a 2×2 matrix exponential as in [31] was used for the CFQM exponential integrators.

As we are interested in studying the complexity-accuracy trade-off of the NFT algorithms, we need a measure of computational complexity. In the literature, either number of floating point operations (FLOPs) or execution times are used as a measure of the computational complexity. Both are not ideal. Although FLOP counting seems more objective, in practice FLOP counts are (just like execution times) implementation specific and it is hard to determine even the number of FLOP counts of basic operations such as square roots. FLOP counts also do not account for typical capabilities of modern processors and neglect critical issues such as memory access. We will present our results in terms of execution times as we believe that they are more realistic than FLOP counts. However, to ensure that our implementations were equally efficient, we carried out an additional FLOP count analysis in Appendix A. By comparing the FLOP counts with the measured execution times we show there that the measured execution times agree well with the FLOP counts for medium to high number of samples. We also show there that the FLOP counts are not representative of computation costs for low number of samples.

Execution times were recorded using the MATLAB stopwatch function (tic-toc). We report the best execution time among three runs to ensure that we minimize the impact of unrelated background processes.
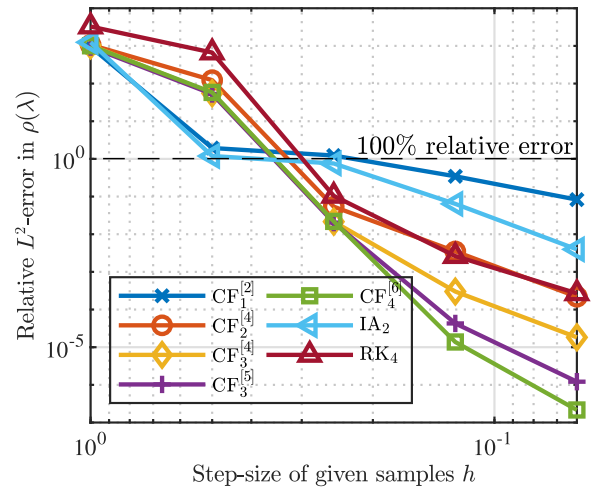

FIGURE 1. Error using slow NFT algorithms for Example 1 with $\lambda_{\max} = 1$.

Fig. 1 shows the error measure E_ρ, as defined in (12), for Example 1 for a range of relatively large step-sizes h. To read such error plots we look at the error achieved by each method for a particular step-size. For the largest two step-sizes, all the errors are above 100 percent and hence a comparison of the methods is not meaningful. The remaining results suggest that the higher-order methods can always be preferred over the lower-order methods.
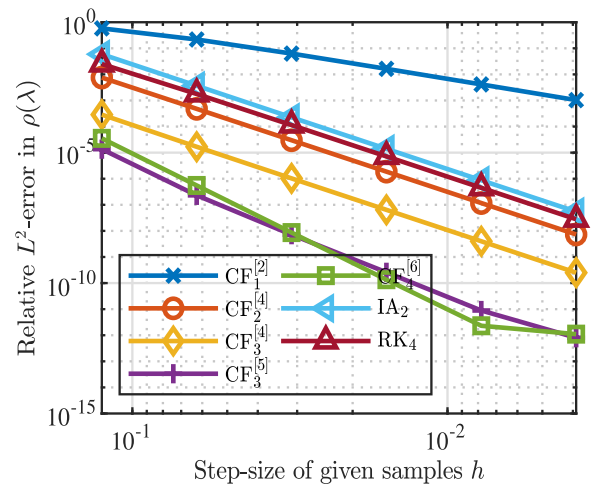

FIGURE 2. Error using slow NFT algorithms for Example 1 with $\lambda_{\max} = 10$.

The error measure E_ρ for smaller sampling intervals h for Example 1, 2, and 3 are shown in Figs. 2, 3 and 4 respectively.[4] For all three examples, the slopes of the error-lines are in agreement with the order r of each method except for IA$_2$. For smooth signals IA$_2$ is seen to have an error of order four rather than the expected three. This observation is in agreement with [13, Fig. 2]. However, for the discontinuous

[4] To ensure that the discontinuity in Example 2 is faithfully captured, we use $t_n = T_- + nh$ for the Runge-Kutta method and the Implicit-Adams method, instead of the description in Section III-A.
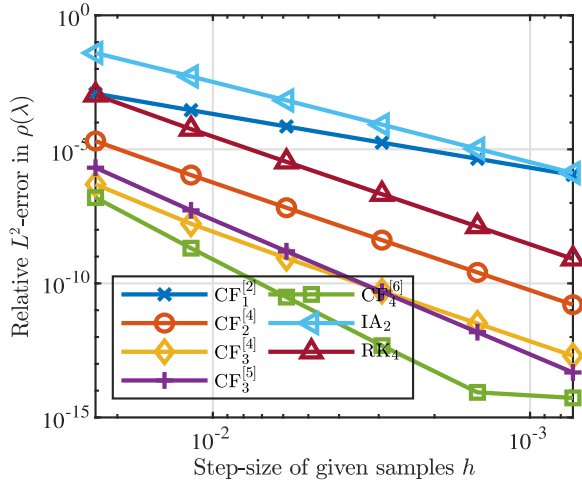
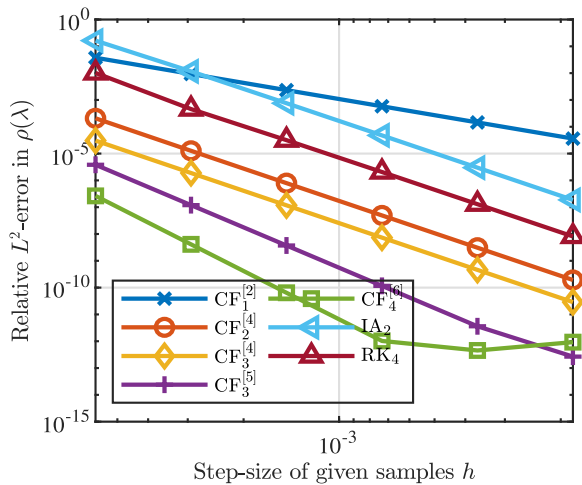**FIGURE 3.** Error using slow NFT algorithms for Example 2.

FIGURE 4. Error using slow NFT algorithms for Example 3.

signal of Example 2 we see third-order behavior as expected. We can also see that a higher r generally corresponds to better accuracy (lower E_ρ) for the same h. However, that is not necessarily obvious as seen in Fig. 2, where $CF_3^{[5]}$ is more accurate than $CF_4^{[6]}$ for larger h. The advantage of using three exponentials ($J = 3$) in $CF_4^{[3]}$ instead of two in $CF_4^{[2]}$ is also clear from the figures. The third-order Implicit-Adams method (IA$_2$ with $r = 3$) and fourth-order Runge-Kutta method (RK$_4$) may be more accurate than $CF_2^{[1]}$ depending on the signal and other parameters, but have lower accuracy compared to $CF_4^{[2]}$ and $CF_4^{[3]}$.

The error E_ρ reaches a minimum around 10^{-12} and can start rising again as seen in Fig. 4 for $CF_6^{[4]}$. To understand this behavior, note that the local error in (8) is actually $\mathcal{O}(h^{r+1} + \varepsilon)$, where ε is a small constant due to finite precision effects that can normally be neglected. The global error is thus $\mathcal{O}(h^r + \varepsilon h^{-1})$. As h is becoming smaller and smaller, the second component also known as the arithmetic error, becomes dominant and eventually causes the total error to rise again [32].
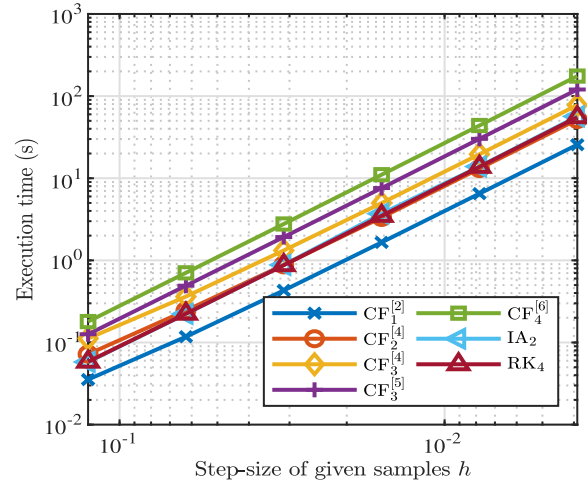

FIGURE 5. Execution time using slow NFT algorithms for Example 1.

For the CFQM exponential integrators, computation of the transfer-matrix $H(\lambda)$ in (7) for each λ requires JD multiplications of 2×2 matrices (8) for $D(\propto 1/h)$ given samples. If the reflection coefficient is to be computed at D points then the overall computational complexity will be of the order $\mathcal{O}(D^2)$. In Fig. 5 we plot the execution times of all the methods for Example 1. These execution times are representative for all examples. We can see that the execution time scales quadratically with $1/h$. The execution time of the CFQM exponential integrators is approximately a linear function of J. The IA$_2$, RK$_4$ and $CF_4^{[2]}$ methods have similar execution times. Although both $CF_4^{[3]}$ and $CF_5^{[3]}$ methods require 3 matrix exponentials, the execution times of $CF_5^{[3]}$ are higher because it involves more operations using complex numbers compared to $CF_4^{[3]}$.
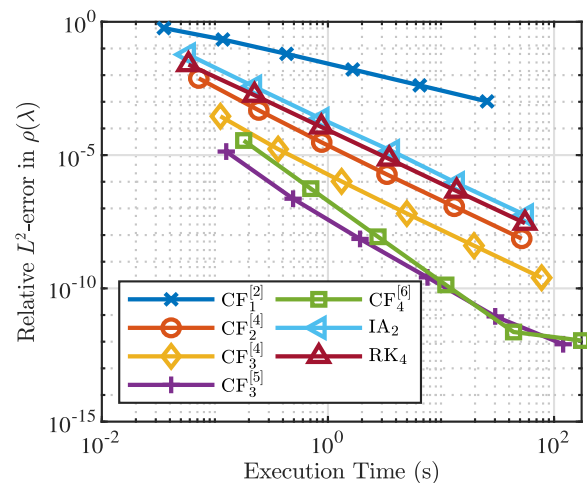

FIGURE 6. Error vs. Execution time trade-off using slow NFT algorithms for Example 1.

To evaluate the trade-off between the execution time and accuracy, we plot the execution time on the x-axis and the error on the y-axis in Fig. 6 for Example 1. To read such

trade-off plots we look at the error achieved by each method for a given amount of time. For Example 1 it turns out that $CF_5^{[3]}$ provides the best trade-off, but we can conclude that extra computation cost of the higher-order methods is generally justified by increased accuracy.

Although performing matrix multiplications of 2×2 matrices is fast, the total cost of the NFT ($\mathcal{O}(D^2)$) is significantly higher when compared to its linear analogue, the FFT, which has a complexity of only $\mathcal{O}(D \log D)$. So the natural question to ask is: Can the complexity be reduced? – Yes, this will be shown in the next section.

IV. FAST FOURTH-ORDER NFT

In this section we investigate which of the new higher-order NFT algorithms from the previous section can be made fast by using suitable splittings of the matrix exponential. **We find that only the $CF_4^{[2]}$ NFT can be made fast. The result is a fast fourth-order NFT algorithm. We then compare the slow $CF_4^{[2]}$ NFT with its fast variant to ensure that the gain in computational complexity outweighs the loss in accuracy introduced by approximations of the matrix exponential.**

A. FAST SCATTERING FRAMEWORK

In the framework proposed in [8], each matrix $G_n(\lambda)$ is approximated by a rational function matrix $\hat{G}_n(z)$, where $z = z(\lambda)$ is a transformed coordinate. By substituting these approximations in (7), a rational function approximation $\hat{H}(z)$ of $H(\lambda)$ is obtained.

$$\hat{H}(z) = \prod_{n=0}^{D-1} \hat{G}_n(z). \tag{21}$$

We want to compute the coefficients of the numerator and denominator polynomials, respectively. A straightforward implementation of the matrix multiplication where each entry is a polynomial, has a complexity of $\mathcal{O}(D^2)$. Instead, by using a binary-tree structure and FFTs [8, Alg. 1], the computational complexity can be reduced to $\mathcal{O}(D \log^2 D)$. Hence it is referred to as fast scattering. In [8], the number of samples D was assumed to be a power of two. In cases where D is not a power of two, we use the following approach. We write $D = 2^{D_1} + 2^{D_2} + \ldots + 2^{D_m}$, where $D_1, D_2, \ldots, D_m$ are non-negative integers. We first choose D_1 as large as possible. Then we choose D_2 as large as possible and repeat until all D_k are fixed. This step splits the D samples into m sets to each of which the fast scattering is applied. The results $\hat{H}_1(z), \hat{H}_2(z) \ldots \hat{H}_m(z)$ are then multiplied using the rule $\hat{H}(z) = [\ldots [[\hat{H}_1(z)\hat{H}_2(z)]\hat{H}_3(z)] \ldots]\hat{H}_m(z)$. Each multiplication is carried out using the same FFT based algorithm as in [8].

The rational function approximation $\hat{H}(z)$ is explicitly parametrized in z and hence (7) is reduced to polynomial evaluations for each z. To elaborate, we again restrict ourselves to $G_n(\lambda)$ of the form (8). Hence for $CF_1^{[2]}$, we need to approximate $G_n(\lambda) = \text{expm}(hC_n(\lambda))$. The matrix exponential can

be approximated to various orders of accuracy using rationals [33] or using splitting-schemes such as the well-known Strang-splitting and the higher-order splitting-schemes developed in [21]. The splitting-schemes map $\lambda \in \mathbb{R}$, the domain of the reflection coefficient, to $z = \exp(i\lambda h/m)$ on the unit circle, where m is a real rational. Such mappings have a certain advantage when it comes to polynomial evaluations which we cover in Section IV-B. Note that the mapping $z = e^{i\lambda h/m}$ is periodic in λ with period $2\pi m/h$. Hence we can resolve the range $|\text{Re}(\lambda)| < \pi m/h$. (See e.g. [34].) This is similar to the Nyquist–Shannon sampling theorem for the FFT.

For a higher-order $CF_J^{[r]}$ integrator, each $G_n(\lambda)$ is a product of J matrix exponentials. For example let us look at $CF_2^{[4]}$. We can write

$$G_n(\lambda) = \text{expm}(hC_n^2(\lambda))\, \text{expm}(hC_n^1(\lambda)),$$
$$C_n^2(\lambda) = a_2 C(T_- + (n + c_1)h, \lambda) + a_1 C(T_- + (n + c_2)h, \lambda),$$
$$C_n^1(\lambda) = a_1 C(T_- + (n + c_1)h, \lambda) + a_2 C(T_- + (n + c_2)h, \lambda),$$
$$a_1 = \frac{1}{4} + \frac{\sqrt{3}}{6}, \quad a_2 = \frac{1}{4} - \frac{\sqrt{3}}{6},$$
$$c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}, \quad c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}. \tag{22}$$

Each of the two matrix exponentials can be approximated individually using a splitting-scheme from [21]. $\hat{H}(z)$ can then be obtained as in (21). However, there are a few caveats which prevent extension of this idea to higher-order methods. The splitting-schemes in [21] should not be applied to CFQM exponential integrators with complex coefficients a_{jk}. Complex coefficients mean that $\lambda \in \mathbb{R}$ is no longer mapped to z on the unit circle. Such a mapping is undesirable for polynomial evaluation as will be explained in Section IV-B. In addition, we do not even obtain a polynomial structure if there exists no z such that $\exp(i\lambda h \sum_{k=1}^{K} a_{j,k})$ is an integer power of this z for all j. Furthermore, if such a z exists but only for high co-prime integer powers, $\hat{G}_n(\lambda)$ will consist of sparse polynomials of high degree, which can significantly hamper the computational advantage of using the approximation. Due to these reasons we restrict ourselves to fast implementations of $CF_1^{[2]}$ and $CF_2^{[4]}$ which will be referred to as $FCF_1^{[2]}$ and $FCF_2^{[4]}$. Even though we made the $FCF_1^{[2]}$ algorithm available in the FNFT software library [35] already, accuracy and execution times for it haven't been assessed and published formally anywhere in literature. The $FCF_2^{[4]}$ algorithm is completely new. For both $FCF_1^{[2]}$ and $FCF_2^{[4]}$ we use the fourth-order accurate splitting [21, Eq. 20].

B. FAST EVALUATION

Once we obtain the rational function approximation $\hat{H}(z)$ of $H(\lambda)$ in terms of numerator and denominator coefficients, we only have to evaluate the numerator and denominator polynomials for each value of $z = z(\lambda)$ in order to compute the reflection coefficient. The degree of the polynomials to be evaluated will be at least D which can be in the range of 10^3–10^4. It is known that evaluation of such high-degree

polynomials for large values of z can be numerically problematic [9, Section IV-E]. However, by choosing the mapping $z = \exp(i\lambda h/m)$, which maps the real line to the unit circle, the polynomials need to be evaluated on the unit circle where evaluation of even high-degree polynomials is numerically less problematic. The higher-order splitting schemes in [21] were developed with such a mapping in mind allowing for approximations of the matrix exponentials as rational functions in z. Evaluating any polynomial of degree D using Horner's method has a complexity of $\mathcal{O}(D)$ [9, Section IV-E]. Hence for M values of z, the total cost of fast scattering followed by polynomial evaluation would be $\mathcal{O}(D\log^2 D) + \mathcal{O}(MD)$.

Mapping $\lambda \in \mathbb{R}$ to z on the unit circle has an additional computational advantage. Let $p(z) = p_N z^N + p_{N-1} z^{N-1} + \ldots + p_0$ be a polynomial in z of degree N. Evaluation of $p(z)$ at a point z_k can be written as

$$p(z_k) = \sum_{n=0}^{N} p_n z_k^n = z_k^N \sum_{n=0}^{N} p_{N-n} z_k^{-n}. \tag{23}$$

For M equispaced points z_k, $k = 1, \ldots, M$, on a circular arc, this amounts to taking the chirp Z-transform (CZT) of the polynomial coefficients. The CZT can be computed efficiently using the algorithm in [36] which utilizes FFTs. We can also see (23) as a non-uniform discrete Fourier transform of the polynomial coefficients which allows us to utilize efficient non-uniform FFT (NFFT) algorithms in [37] for evaluating the polynomial. If the number of evaluation points M is in the same order of magnitude as D, the complexity of evaluation becomes $\mathcal{O}(D\log D)$ and hence the overall complexity of the fast nonlinear Fourier transform (FNFT) is $\mathcal{O}(D\log^2 D)$. In the next section we will see that the error of the $\mathrm{FCF}_2^{[4]}$ algorithm reaches a minimum value and then starts rising. This is again due to the arithmetic error as we already saw in Section III-D. We remark that in numerical tests the CZT was found to perform equally well as the NFFT before the error minimum but the error rise thereafter was significantly steeper. We hence used the NFFT routine from [37] for evaluating the polynomials.

C. NUMERICAL EXAMPLES
1) REFLECTION COEFFICIENT
We now compare the implementations of $\mathrm{CF}_1^{[2]}$ and $\mathrm{CF}_2^{[4]}$ presented in Section III-D and their fast versions $\mathrm{FCF}_1^{[2]}$ and $\mathrm{FCF}_2^{[4]}$ for computing the reflection coefficient $\rho(\lambda)$. We plot the error versus the execution time for Example 1 in Fig. 7, for Example 2 in Fig. 8 and for Example 3 in Fig. 9. In the three figures we can see that the fast FCF algorithms achieve similar errors as their slow CF counterparts in a significantly shorter time. From the other viewpoint, for the same execution time, the FCF algorithms achieve significantly lower errors compared to CF algorithms. Our new algorithm $\mathrm{FCF}_2^{[4]}$ outperforms $\mathrm{FCF}_1^{[2]}$ in the trade-off for all the examples which again highlights the advantage of using higher-order CFQM exponential integrators.
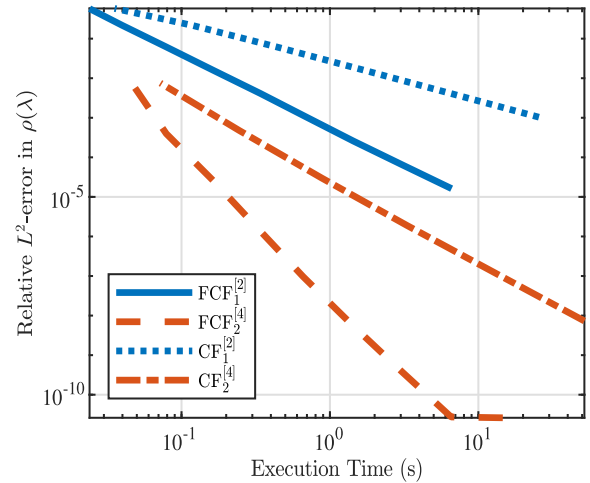

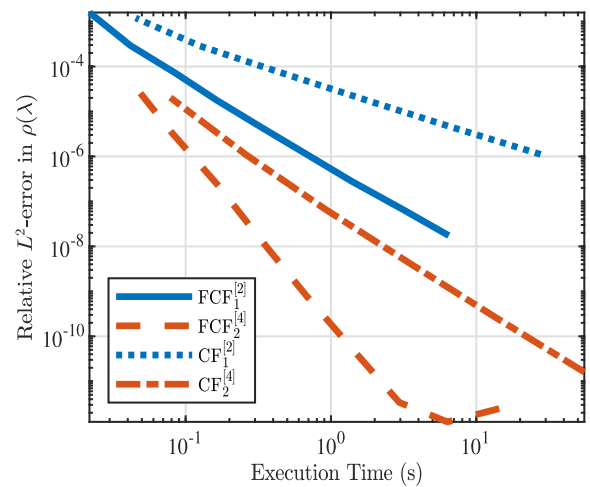**FIGURE 7.** Error vs. Execution time trade-off using CF and FCF algorithms for Example 1.

FIGURE 8. Error vs. Execution time trade-off using CF and FCF algorithms for Example 2.
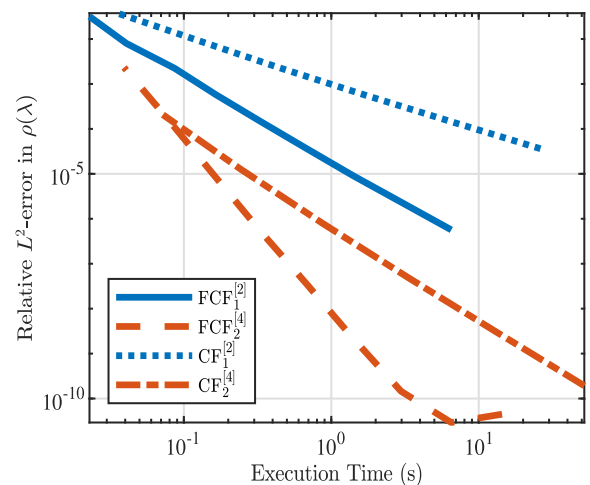

FIGURE 9. Error vs. Execution time trade-off using CF and FCF algorithms for Example 3.

Since the NFT is a nonlinear transform, it changes its form under signal amplification, and computing it typically becomes increasingly difficult when a signal is
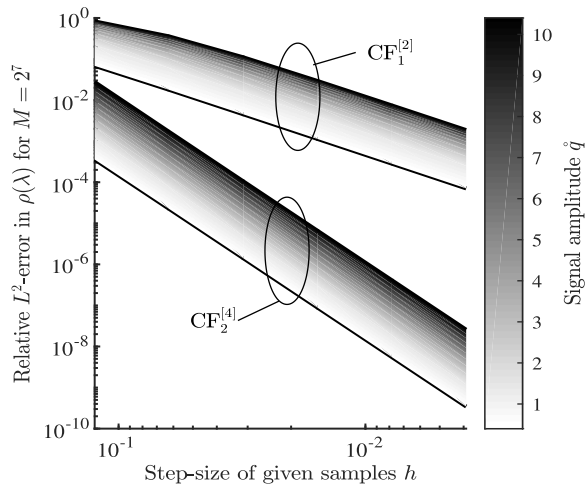
FIGURE 10. Variation of error of CF algorithms with amplitude for Example 1. The fourth-order $CF_2^{[4]}$ algorithm is seen to have gradual increase in error with increase in amplitude similar to the second-order $CF_1^{[2]}$ algorithm.
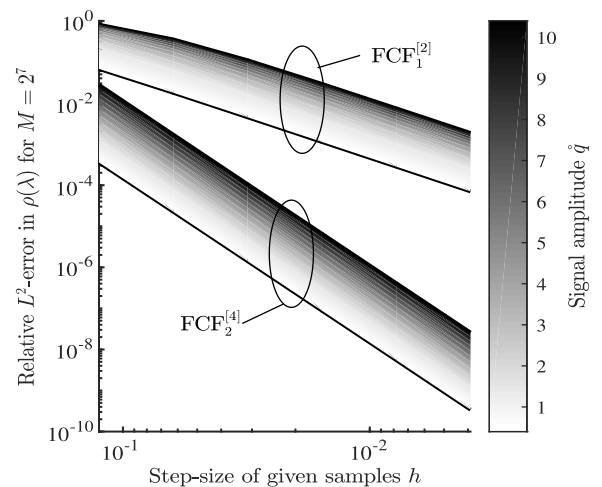

FIGURE 11. Variation of error of FCF algorithms with amplitude for Example 1. Approximating the matrix exponentials with splitting schemes does not significantly affect the amplification of error with increasing amplitude.

amplified [22]. Hence it is of interest to study amplification of error with increase in signal amplitude. To test the amplification we use Example 1 and sweep the signal amplitude $\mathring{q}$ from 0.4 to 10.4 in steps of 1.0 while keeping all other parameters the same as before. As the time-window remains the same, amplification the signal amplitude leads to directly proportional amplification of the signal energy. We compute the error E_ρ for decreasing h for each value of $\mathring{q}$ for the CF and FCF algorithms. We plot E_ρ versus the sampling interval h on a log-scale for CF algorithms in Fig. 10 and for FCF algorithms in Fig. 11. Instead of plotting individual lines for each value of $\mathring{q}$, we represent the amplitude using different different shades of gray. As shown in the colourbar, lighter shades represents lower $\mathring{q}$ and darker shades represent higher $\mathring{q}$. The stripes with a higher slope are the higher-order methods. All the four algorithms i.e., $CF_1^{[2]}$, $CF_2^{[4]}$, $FCF_1^{[2]}$ and $FCF_2^{[4]}$ show similar trends for the amplification of error with signal amplitude. The $CF_1^{[2]}$ algorithm was compared with other methods in [22] (where it is referred to as BO), and they conclude that $CF_1^{[2]}$ scales the best with increasing signal amplitude. Hence the results shown in Fig. 10 are very motivating as the amplification in the error of $CF_2^{[4]}$ is similar to the amplification for $CF_1^{[2]}$. The error of approximations used in the FCF algorithms also depends on $\mathring{q}$. However, comparing Fig. 10 and Fig. 11 we can see that the contribution of the approximation error is small. These results combined with the results in the trade-off plots (Figs. 7, 8, and 9) make a strong case for our new $FCF_2^{[4]}$ algorithm.

2) B-COEFFICIENT
The accurate and fast computation of the scattering coefficient $b(\lambda)$ (Section II) is of interest to the fiber-optic communication community, as an efficient FNFT algorithm can be combined with the recently proposed b-modulation
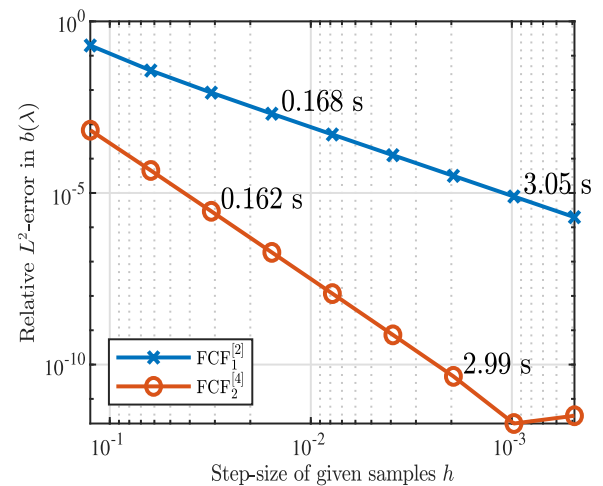
FIGURE 12. Error in b-coefficient using FCF algorithms for Example 1. The execution times for some of the points are shown to give an indication of the computational complexity.

[38]–[40] scheme to develop a complete NFT based fiber-optic communication system. Hence to test the performance of both the FCF algorithms in computation of the b-coefficient, we define

$$E_b = \frac{\sqrt{\sum_{n=0}^{M-1} |b(\lambda_n) - \hat{b}(\lambda_n)|^2}}{\sqrt{\sum_{n=0}^{M-1} |b(\lambda_n)|^2}}, \qquad (24)$$

where $b(\lambda)$ is the analytically known and $\hat{b}(\lambda)$ is the numerically computed scattering coefficient. For the numerical test we again use the signal from Example 1 as $b(\lambda)$ is known. We plot the error E_b for both the FCF methods for decreasing sampling interval h in Fig. 12. $FCF_2^{[4]}$ clearly outperforms $FCF_1^{[2]}$ even after considering the additional computational cost.

From the results of the numerical tests presented in this section it is clear that approximating $H(\lambda)$ in (7) using rational functions to make the method fast, provides a significant computational advantage: similar accuracy, shorter execution time. However, as mentioned earlier we could only make the fourth-order method $\text{CF}_2^{[4]}$ fast. To further improve the accuracy and order of convergence while restricting ourselves to a fourth-order method, we explore the possibility of using Richardson extrapolation in the next section.

V. MAIN RESULT: FAST SIXTH-ORDER NFT

In this section we arrive at our main result by integrating Richardson extrapolation into our new fast fourth-order NFT $\text{FCF}_2^{[4]}$ from the previous section. **We show numerically that the resulting algorithm has *sixth-order* accuracy rather than fifth-order as would be expected. We furthermore show that the added complexity due to Richardson extrapolation is outweighed by the gain in accuracy so the complexity-accuracy trade-off of our final algorithm is the best among all methods investigated in this paper.**

A. RICHARDSON EXTRAPOLATION

Richardson extrapolation is a technique for improving the rate of convergence of a series [41].[5] Given an rth-order numerical approximation method $\hat{\rho}(\lambda, h)$ for the reflection coefficient $\rho(\lambda)$ that depends on the step-size h, we can write

$$\rho(\lambda) = \hat{\rho}(\lambda, h) + \mathcal{O}(h^r). \tag{25}$$

We assume that $\hat{\rho}(\lambda, h)$ has a series expansion in h,

$$\hat{\rho}(\lambda, h) = \rho(\lambda) + \rho_r(\lambda)h^r + \rho_{r+1}(\lambda)h^{r+1} + \dots \tag{26}$$

In Richardson extrapolation [41], we combine two numerical approximations $\hat{\rho}(\lambda, h)$ and $\hat{\rho}(\lambda, 2h)$ as follows,

$$\hat{\rho}^{[\text{RE}]}(\lambda, h) = \frac{2^r \hat{\rho}(\lambda, h) - \hat{\rho}(\lambda, 2h)}{2^r - 1}. \tag{27}$$

Using the series expansion, we find that the order of the new approximation $\hat{\rho}^{[\text{RE}]}(\lambda, h)$ is at least $r + 1$:

$$\hat{\rho}^{[\text{RE}]}(\lambda, h) = \frac{2^r(\rho(\lambda) + \rho_r(\lambda)h^r + \rho_{r+1}(\lambda)h^{r+1} + \dots)}{2^r - 1}$$
$$- \frac{\rho(\lambda) + \rho_r(\lambda)(2h)^r + \rho_{r+1}(\lambda)(2h)^{r+1} + \dots}{2^r - 1}$$
$$= \rho(\lambda) - \frac{2^r}{2^r - 1}\rho_{r+1}(\lambda)h^{r+1} + \mathcal{O}(h^{r+2}). \tag{28}$$

We apply this idea to $\text{FCF}_1^{[2]}$ and $\text{FCF}_2^{[4]}$ to obtain the algorithms $\text{FCF\_RE}_1^{[2]}$ and $\text{FCF\_RE}_2^{[4]}$ respectively. Note that the range of $|\text{Re}(\lambda)|$ that can be resolved is determined by the larger of the two step-sizes h (see Section IV-A). We also remark that Richardson extrapolation can also be applied to the slow algorithms in Section III-C.

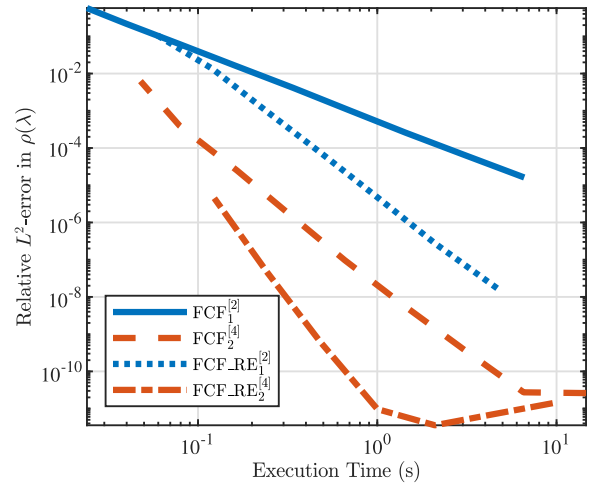[5]It was used to improve an inverse NFT algorithm for the defocusing case in [42].

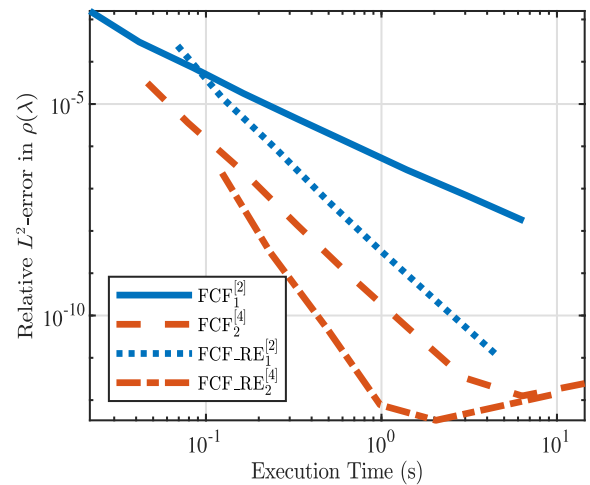**FIGURE 13.** Error vs. Execution time trade-off of FCF and FCF_RE algorithms for Example 1.

FIGURE 14. Error vs. Execution time trade-off of FCF and FCF_RE algorithms for Example 2.

B. NUMERICAL EXAMPLES

We test $\text{FCF\_RE}_1^{[2]}$ and $\text{FCF\_RE}_2^{[4]}$ against $\text{FCF}_1^{[2]}$ and $\text{FCF}_2^{[4]}$ for all three examples. Since Richardson extrapolation requires us to compute two approximations, which increases the computational complexity, we again evaluate the complexity-accuracy trade-off. We plot the error versus execution time curves for the three examples in the Figs. 13 to 15. In all figures we can see that the FCF_RE algorithms achieve slopes of $r + 2$ rather than the expected slope of $r + 1$. This is an example of superconvergence [43]. Specifically, the error of $\text{FCF\_RE}_1^{[2]}$ decreases with slope four and that of $\text{FCF\_RE}_2^{[4]}$ decreases with slope six. As seen before in Section III-D, the arithmetic error starts to dominate after a certain point and causes the error to rise. Although the executions times of FCF_RE algorithms are higher for the same step-size h, the error achieved is almost an order of magnitude lower even for large h. From the other viewpoint, for the same execution time, the FCF_RE algorithms achieve significantly lower errors compared to
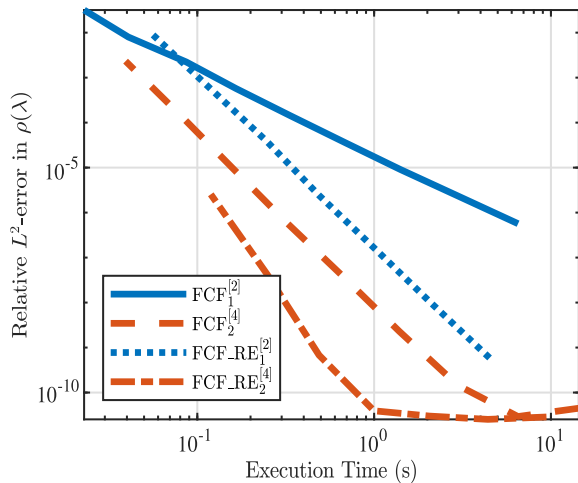
FIGURE 15. Error vs. Execution time trade-off of FCF and FCF_RE algorithms for Example 3.

FCF algorithms. $\text{FCF\_RE}_2^{[4]}$ outperforms $\text{FCF\_RE}_1^{[2]}$ in the trade-off for all the three examples again highlighting the advantage of using higher-order CFQM exponential integrators. These results suggest that Richardson extrapolation should be applied to improve the considered FNFT algorithms. The $\text{FCF\_RE}_2^{[4]}$ algorithm provides the best trade-off among all the algorithms presented in this paper.

C. REMARKS ON COMPUTING EIGENVALUES

The main focus of this paper has been the efficient computation of the reflection coefficient. The computation of the discrete spectrum (see Section II) is more involved and problem specific. The best approach strongly depends on the available a priori knowledge on the number and location of the eigenvalues. In scenarios where little a priori knowledge is available, some of the ideas presented for the reflection coefficient can be applied to the discrete spectrum as well. Some possible approaches are discussed in Appendix C.

VI. CONCLUSION

In this paper, we proposed new higher-order nonlinear Fourier transform algorithms based on a special class of exponential integrators. We also showed that one of these algorithms can be made fast using special higher-order exponential splittings. The accuracy of the fast algorithm was improved even further, to sixth-order, using Richardson extrapolation. **To the best of our knowledge this is the first fast sixth-order NFT algorithm ever presented in the literature.** Numerical demonstrations showed that the proposed algorithm is highly accurate and provides much better complexity-accuracy trade-offs than existing algorithms. In the future we plan to integrate the algorithms from this paper into the open source software library FNFT [35]. We finally remark that the development of a fast higher-order inverse NFT is an interesting open topic for future research.

TABLE 1. Number of operations per type.

Operation	Algorithm	
	$\text{CF}_2^{[4]}$	$\text{FCF}_2^{[4]}$
FFT (no, sz)	$3, D$	$3, D$
Multiplication	$14D + 9M + 18DM$	$86D + 4(M+1)$
Addition	$4(D+1) + 10DM$	$34D + 6$
Division	$2DM$	$24D + M$
Conjugation	$2D$	$2D$
Square-root	$2DM$	$2D$
sinh	$2DM$	-
cosh	$2DM$	-
cos	-	$4D$
sinc	-	$4D$
Exponential	$2M$	$2M$
CZT (no, sz)	-	$2, 4D+1$
Fast scattering (no,sz)	-	$1, 2D$

The abbreviations no and sz are short for number and size respectively. All operations are assumed to be on complex numbers. The number of signal samples (D) is assumed to be greater than the number of reflection coefficient samples (M) being computed, i.e. $D \geq M$.

TABLE 2. Number of FLOPs for various operations.

Operation	Number of FLOPs
FFT of size N	$5N \log_2 N$
Multiplication	1
Addition	1
Division	4
Conjugation	1
Square-root	4
sinh or cosh	8
sin or cos	8
sinc	12
Exponential	8
CZT of size N	$3(5(2N-1)\log_2(2N-1))$
Fast scattering of size N	See (29)

The number of FLOPs for the basic operations have been taken from [44, p. 5]. The number of FLOPs for a sinc are the sum of number of FLOPs for a sin and a division. The number of FLOPs for a FFT are based on the asymptotic number of operations for the radix-2 Cooley-Tukey algorithm [45, p. 3]. The number of FLOPs for a CZT are approximated using three FFTs of size $2N - 1$.

APPENDIX A
COMPARISON OF FLOP COUNTS AND EXECUTION TIMES

In this section we show a comparison between the number of floating-point operations (FLOPs) and execution times of two algorithms proposed in this paper. We counted all the operations of the slow algorithm and the fast algorithm based on the $\text{CF}_2^{[4]}$ integrator. We list the different operations and how often they occur in Table 1. For the FFT and CZT, the size of the input is also specified. The number of FLOPs required for each operation type are provided in Table 2. Note that these values are only rules of thumb and vary widely across programming languages and CPU architectures. The number of FLOPs required for the fast scattering step (see Section IV-A) is given by

$$\text{FLOPs(Fast scattering of size N)}$$
$$= \sum_{k=0}^{\lceil \log_2 N \rceil} 2^{\lceil \log_2 N \rceil - k} \Big(12\,\text{FLOPs(FFT of size}(2^{k+1}+1))$$
$$+ (8\,\text{FLOPs(Mult)} + 4\,\text{FLOPs(Add)})(2^{k+1}+1) \Big). \quad (29)$$
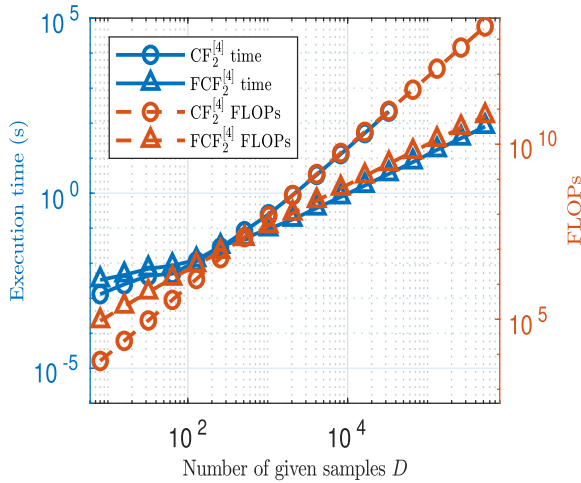
FIGURE 16. The number of FLOPs and measured execution times of a slow fourth-order algorithm and its fast variant.

In Fig. 16, we plot the total number of FLOPs and the execution times from our MATLAB implementations against the number of given samples D. At medium to high number of samples we see that the MATLAB execution times match the number of FLOPs very well. Moreover the crossover point at which the fast algorithm becomes faster than the slow variant ($D > 300$ in Fig. 16) is almost the same. At lower number of samples, the execution times deviate from the number of FLOPs. This is due to the unaccounted overheads dominating over the floating-point operations.

APPENDIX B
INTERPOLATION BASED ON FOURIER TRANSFORM

```
function qs = bandlimited_interp(tn,qn,
    ts)
% Inputs
% tn - Sorted vector of equispaced
    points at
%      which qn samples are known
% qn - Vector of signal samples at
%      points tn
% ts - Value by which samples are
%      to be shifted
% Output
% qs - Vector of interpolated signal
%      samples at tn-ts
ep = tn(2)-tn(1);
Qn = fft(qn);
N = length(qn);
Np = floor(N/2);
Nn = -floor((N-1)/2);
Qn = Qn.*exp(2i*pi*[0:Np,Nn:-1]*ts/(N*ep
    ));
qs = ifft(Qn);
```

APPENDIX C
COMPUTING EIGENVALUES

Recall that for the case of focusing NSE ($\kappa = 1$), the non-linear Fourier spectrum has two parts: a continuous and

a discrete part. In this appendix, we are concerned with the numerical computation of the discrete part. We first mention some of the existing approaches and then show how one of them can be extended to work with the new fast higher-order NFT algorithms. We will also show that Richardson extrapolation can be applied to improve the accuracy at virtually no extra computational cost.

A. EXISTING APPROACHES
Finding the eigenvalues consists of finding the complex upper half-plane roots of the function $a(\lambda)$. Most of the existing approaches can be classified into four main categories.

1) *Search methods:* Newton's method.
2) *Eigenmethods:* Spectral methods based on the solution of a suitably designed eigenproblem [27].
3) *Gridding methods:* They find λ_k using iterative methods or optimized grid search [11], [27]. Recently a method based on contour integrals was proposed [46].
4) *Hybrid methods:* Any combination of the above. Eigenmethods with rougher sampling can e.g. be used to find initial guesses for a search method [47].

Our proposed method will be a hybrid of a eigenmethod and a search method in the spirit of [47], where an eigenproblem is solved to obtain initial guesses for Newton's method.

B. PROPOSED METHOD
Remember that the discrete spectrum consists of eigenvalues, which are the zeros of $a(\lambda)$ in the complex upper half-plane ($\mathbb{H}$), and their associated residues. We start with discussing an approximation of $a(\lambda)$ that will be useful for locating the eigenvalues. From (3), (4) and (5) we can write

$$a(\lambda) = \lim_{t \to \infty} \phi_1(t, \lambda)e^{i\lambda t}. \qquad (30)$$

Over the finite interval $[T_-, T_+]$ using (7) we can see that

$$a(\lambda) \approx H_{1,1}(\lambda)e^{i\lambda T_-}e^{i\lambda T_+}. \qquad (31)$$

Hence we hope that the zeros of $H_{1,1}(\lambda)$ are approximations of the zeros of $a(\lambda)$ if the signal truncation and discretization errors are small enough. In Section IV-A we explained how $H(\lambda)$ can be approximated by a rational function in a transformed coordinate z. Hence we can further write

$$a(\lambda) \approx \frac{\hat{a}_{\text{num}}(z)}{\hat{a}_{\text{den}}(z)}e^{i\lambda T_-}e^{i\lambda T_+}, \qquad (32)$$

where $\hat{a}_{\text{num}}(z)$ and $\hat{a}_{\text{den}}(z)$ are polynomials in $z(\lambda)$. Let $\hat{a}_{\text{num}}(z) = \hat{a}_N z^N + \hat{a}_{N-1}z^{N-1} + \ldots + \hat{a}_0$. Thus $\hat{a}_{\text{num}}(z)$ will have N zeros. These zeros or roots of $\hat{a}_{\text{num}}(z)$ can be found using various methods [48]. Of these N zeros, there should be K (typically, $K \ll N$) values that are approximations of zeros of $a(\lambda)$ in $\mathbb{H}$.

EXAMPLE
We would like to add clarity through a visual representation of the roots. We choose the signal from Example 1 with $D = 2^9$. We plot all the zeros of $\hat{a}_{\text{num}}(z)$ of $\text{FCF}_1^{[2]}$ with 'x'
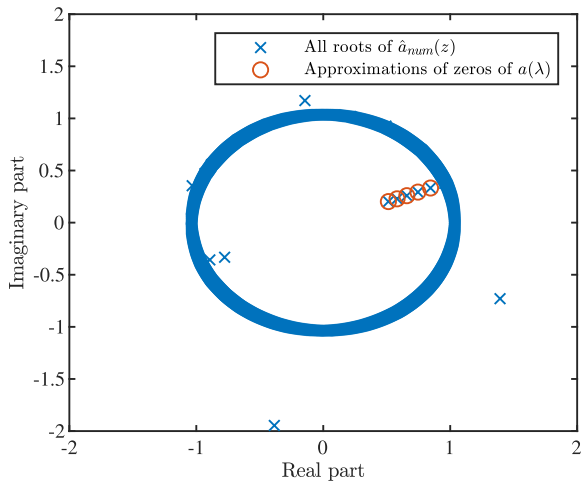
FIGURE 17. Zeros of $\hat{a}_{\mathrm{num}}(z)$ for Example 1 with $D = 2^9$.
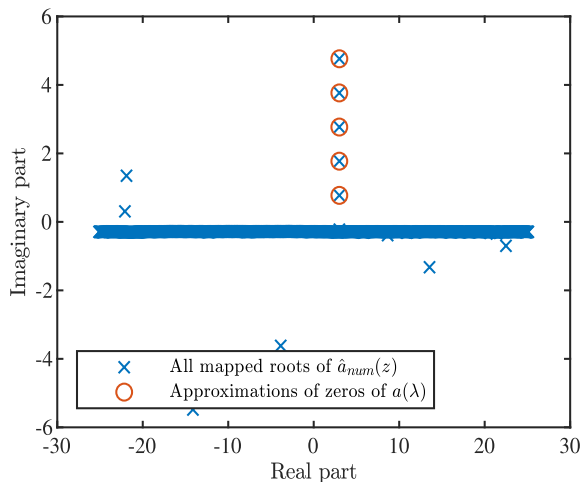

FIGURE 18. Mapped zeros of $\hat{a}_{\mathrm{num}}(z)$ for Example 1 with $D = 2^9$.

in Fig. 17. Here $z = e^{i\lambda h}$. We can then map these zeros back to obtain values of λ. These are plotted with 'x' in Fig. 18. From the definition of discrete spectrum, we can filter out all the values that are not in $\mathbb{H}$. Recall that we can resolve the range $|\mathrm{Re}(\lambda)| < \pi/h$. (See Section IV-A.) Since we observed that spurious eigenvalues tend to cluster around $|\mathrm{Re}(\lambda)| = \pi/h$, we filter out the corresponding roots of $\hat{a}_{\mathrm{num}}(z)$. More precisely we keep only values of λ for which $|\mathrm{Re}(\lambda)| < 0.9\pi/h$. The filtered roots are plotted in Figs. 17 and 18 with 'o'. For the chosen value of $\mathring{q} = 5.4$ the set of eigenvalues is $\Lambda = \{3+4.9i, 3+3.9i, 3+2.9i, 3+1.9i, 3+0.9i\}$. From Fig. 18 we see that the values marked with 'o' are indeed approximations of the values in set Λ. However, there is no guarantee that we will always be able to locate approximations for all values in Λ as that depends on several factors, some of which are the signal magnitude $\mathring{q}_o$, signal interval $[T_-, T_+]$, step-size h and values of the eigenvalues themselves.

For the example chosen in the visual demonstration, the number of zeros is $N = 1024$ and the number of eigenvalues is $K = 5$. For the chosen mapping from $\lambda \to z$,

the K values of interest will always lie inside the unit circle in Fig. 17 and most other spurious zeros of $\hat{a}_{\mathrm{num}}(z)$ will lie on the unit circle. Even with the best eigenmethods available for polynomial root-finding, which have a complexity of $\mathcal{O}(N^2)$ [49], execution time grows very steeply, making this approach infeasible for large N. To reduce the complexity, it was suggested in [47] to sub-sample the given signal to reduce the dimensionality of the root-finding problem. The algorithm is summarized in Fig. 19.

Input: Samples $q_0, \ldots, q_{D-1}$, D_{sub}, $[T_-, T_+]$
Output: Estimated eigenvalues $\hat{\lambda}_k$
1: Subsample the given samples $\to$ $q_n^{\mathrm{sub}} := q_{n\lfloor D/D_{\mathrm{sub}}\rceil}$, where $\lfloor\cdot\rceil$ rounds to the closest integer.
   Build a polynomial approximation $\hat{a}_{\mathrm{num}}(z)$ using the D_{sub} subsampled samples.
   Apply a fast eigenmethod to find the roots of $\hat{a}_{\mathrm{num}}(z)$.
   Filter the roots.
2: Refine the roots via Newton's method using all samples.
   Filter the refined roots.
3: Apply Richardson extrapolation to the unrefined and refined roots.

FIGURE 19. Algorithm : Subsample and refine.

We now discuss the three stages of the algorithm in detail.

1) **Root finding from a subsampled signal**
   The given signal q_n is subsampled to give q_n^{sub} with D_{sub} samples. The corresponding step-size is h_{sub}. There are no results for the minimum number of samples that guarantee that all eigenvalues will be found. One choice can be based on limiting the overall computational complexity to $\mathcal{O}(D\log^2 D)$, which is the complexity for the reflection coefficient. For a root-finding algorithm with $\mathcal{O}(D^2)$ complexity, we choose to use $D_{\mathrm{sub}} = \mathrm{round}\left(\sqrt{D\log^2 D}\right)$ samples. The polynomial $\hat{a}_{\mathrm{num}}(z)$ is then built from these D_{sub} samples. For $\mathrm{FCF}_2^{[4]}$, the non-equispaced samples should be obtained from the original D samples and not the D_{sub} samples. An eigenmethod is then used to find all zeros of $\hat{a}_{\mathrm{num}}(z)$. We used the algorithm in [49]. The values of z are mapped backed to λ and filtered to remove implausible values.

2) **Root refinement using Newton method**
   The Newton method based on the slow CF methods is used for root refinement. The derivative $\mathrm{d}a(\lambda)/\mathrm{d}\lambda$ is calculated numerically along with $a(\lambda)$ as in [14] using all the given samples q_n. The values of λ that remain after filtering in the previous step are used as the initial guesses for the Newton method. We chose to stop the iterations if the change in value goes below 10^{-15} or if a maximum of 15 iterations is reached. The resulting roots are filtered again.

3) **Richardson extrapolation**
   We pair the roots resulting from the Newton step, $\hat{\lambda}_k^{\mathrm{Newton}}$, with the corresponding initial guesses $\hat{\lambda}_k^{\mathrm{init}}$.
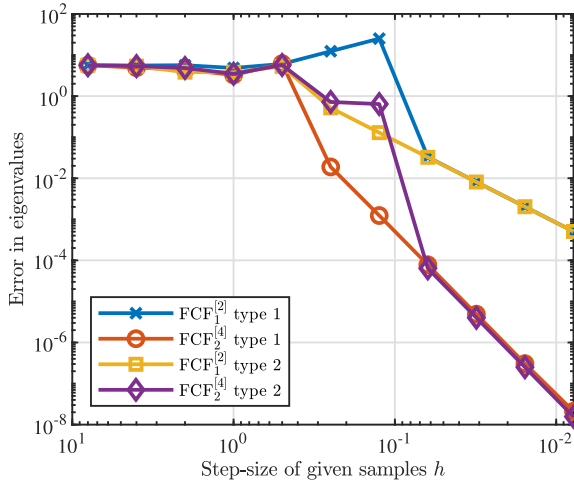
FIGURE 20. Error in approximation of the eigenvalues by the fast second- and fourth-order algorithms of type 1 (no sub-sampling) and type 2 (sub-sample and refine, no Richardson extrapolation).

Then, we apply Richardson extrapolation:

$$\hat{\lambda}_k = \frac{(\frac{h_{\text{sub}}}{h})^r \hat{\lambda}_k^{\text{Newton}} - \hat{\lambda}_k^{\text{init}}}{(\frac{h_{\text{sub}}}{h})^r - 1}. \quad (33)$$

$\hat{\lambda}_k$ is then an improved approximation and constitutes the discrete part of the FCF_RE algorithm. It may so happen that more than one $\tilde{\hat{\lambda}}_k^{\text{init}}$ converge to the same $\hat{\lambda}_k^{\text{Newton}}$. In such cases the value $\hat{\lambda}_k^{\text{init}}$ closest to $\hat{\lambda}_k^{\text{Newton}}$ should be used for Richardson extrapolation. The other values $\hat{\lambda}_k^{\text{init}}$ that also converged to the same $\hat{\lambda}_k^{\text{Newton}}$ should be treated as spurious values and discarded.

The numerical algorithms may not find particular eigenvalues or find spurious ones. Let $\hat{\Lambda}$ be the set of approximations found by an algorithm. To penalize both missed values and incorrect spurious values at the same time, we define the error

$$E\Lambda = \max \left\{ \max_{\lambda_i \in \Lambda} \min_{\hat{\lambda}_j \in \hat{\Lambda}} |\lambda_i - \hat{\lambda}_j|, \max_{\hat{\lambda}_j \in \hat{\Lambda}} \min_{\lambda_i \in \Lambda} |\lambda_i - \hat{\lambda}_j| \right\}. \quad (34)$$

Note that the first term in the outer maximum grows large if an algorithm fails to approximate a part of the set Λ while the second term becomes large if an algorithm finds spurious values that have no correspondence with values in Λ. $E\Lambda$ is expected to be of order r for an algorithm of order r.

C. NUMERICAL EXAMPLE
In this section, we compare different variants of our proposed algorithm using Example 1. We compute the error $E\Lambda$ for the following three types of algorithms:
1) Discrete part of FCF algorithms. An eigenmethod is applied to the approximation $\hat{a}_{\text{num}}(z)$ built using all samples. No sub-sampling is used.
2) Discrete part of FCF algorithms with sub-sampling. Only steps 1 and 2 of the algorithm mentioned above.
3) Discrete part of FCF_RE algorithms. All the three steps mentioned above.

To demonstrate the effect of sub-sampling, we show in Fig. 20 the errors for the second- and fourth-order algorithms of

types 1 and 2. For $h > 0.3$ the errors are high either due to failure to find approximations close to the actual eigenvalues or due to spurious values. For $h \leq 0.3$, $\text{FCF}_2^{[4]}$ of type 1 and $\text{FCF}_1^{[2]}$ of type 2 find exactly five values that are close approximations of the values in Λ. However $\text{FCF}_1^{[2]}$ of type 1 and $\text{FCF}_2^{[4]}$ of type 2 find good approximations only for $h \leq 0.06$. The error of $\text{FCF}_1^{[2]}$ algorithms decreases with slope two and that of $\text{FCF}_2^{[4]}$ algorithms decreases with slope four as expected from the order of the underlying numerical schemes.
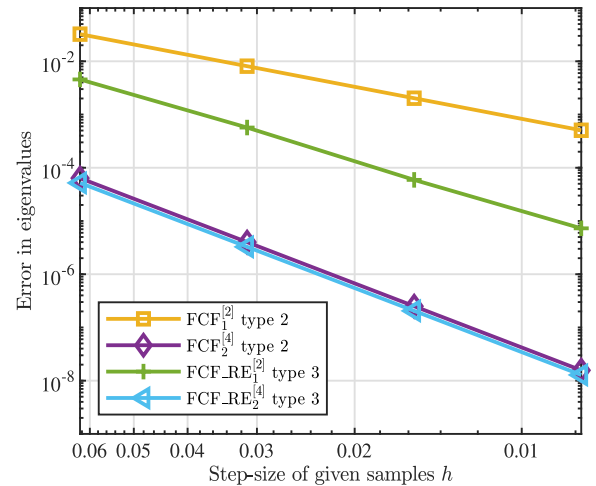

FIGURE 21. Error in approximation of a eigenvalues computed using FCF and FCF_RE algorithms.

In Fig. 21 we show the errors for the second- and fourth-order algorithms of type 2 and 3 to indicate the advantage of the extrapolation step. The extrapolation step improves the approximation significantly for $\text{FCF\_RE}_1^{[2]}$ while adding negligible computation cost to the algorithm. However, there is only minor improvement in case of $\text{FCF\_RE}_2^{[4]}$ over $\text{FCF}_2^{[4]}$.
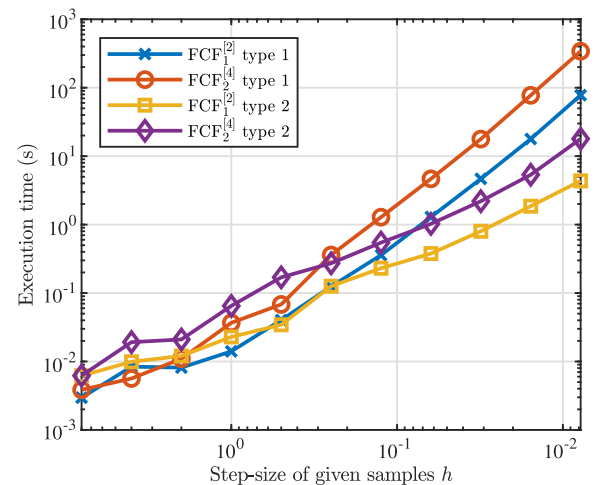

FIGURE 22. Execution time of $\text{FCF}_2^{[4]}$ and $\text{FCF}_2^{[4]}$ algorithms for computing eigenvalues of Example 1.

In Fig. 22 we plot the execution times for the FCF algorithms of types 1 and 2. The execution times of algorithms

of type 3 are almost the same as those of type 2. For type 1 algorithms, these times include the time required to build $\hat{a}_{num}(z)$ and the time taken by the root-finder. For algorithms of type 2, the additional time required for root-refinement by Newton's method is also included. Even with sub-sampling, we see that the execution times are an order of magnitude higher than the execution times for the continuous part. The FCF_RE algorithms seem to provide the best trade-off between accuracy and computation cost similar to the case of continuous part. The overall computational complexity may be decreased by using alternative methods to find the initial guesses.

REFERENCES

[1] C. L. Epstein, "How well does the finite Fourier transform approximate the Fourier transform?" *Commun. Pure Appl. Math.*, vol. 58, no. 10, pp. 1421–1435, 2005.

[2] E. Agrell, M. Karlsson, A. R. Chraplyvy, D. J. Richardson, P. M. Krummrich, P. Winzer, K. Roberts, J. K. Fischer, S. J. Savory, B. J. Eggleton, M. Secondini, F. R. Kschischang, A. Lord, J. Prat, I. Tomkos, J. E. Bowers, S. Srinivasan, M. Brandt-Pearce, and N. Gisin, "Roadmap of optical communications," *J. Opt.*, vol. 18, no. 6, p. 063002, 2016.

[3] S. K. Turitsyn, J. E. Prilepsky, S. T. Le, S. Wahls, L. L. Frumin, M. Kamalian, and S. A. Derevyanko, "Nonlinear Fourier transform for optical data processing and transmission: Advances and perspectives," *Optica*, vol. 4, no. 3, pp. 307–322, 2017.

[4] A. R. Osborne, "Nonlinear Fourier methods for ocean waves," *Procedia IUTAM*, vol. 26, pp. 112–123, Sep. 2017. doi: 10.1016/j.piutam.2018.03.011.

[5] M. Brühl, "Direct and inverse nonlinear Fourier transform based on the Korteweg-deVries equation (KdV-NLFT)—A spectral analysis of nonlinear surface waves in shallow water," Ph.D. dissertation, Dept. Hydromechanics Coastal Eng., Leichtweiß-Inst. Hydraulic Eng. Water Resour., Tech. Univ. Braunschweig, Braunschweig, Germany, Feb. 2014.

[6] N. J. Ablowitz, D. J. Kaup, A. C. Newell, and H. Segur, "The inverse scattering transform-Fourier analysis for nonlinear problems," *Stud. Appl. Math.*, vol. 53, no. 4, pp. 249–315, Dec. 1974.

[7] A. C. Singer, A. V. Oppenheim, and G. W. Wornell, "Detection and estimation of multiplexed soliton signals," *IEEE Trans. Signal Process.*, vol. 47, no. 10, pp. 2768–2782, Oct. 1999. doi: 10.1109/78.790658.

[8] S. Wahls and H. V. Poor, "Introducing the fast nonlinear Fourier transform," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2013, pp. 5780–5784. doi: 10.1109/ICASSP.2013.6638772.

[9] S. Wahls and H. V. Poor, "Fast numerical nonlinear Fourier transforms," *IEEE Trans. Inf. Theory*, vol. 61, no. 12, pp. 6957–6974, Dec. 2015. doi: 10.1109/TIT.2015.2485944.

[10] 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*, vol. 26, no. 13, pp. 17360–17377, 2018.

[11] S. Burtsev, R. Camassa, and I. Timofeyev, "Numerical algorithms for the direct spectral transform with applications to nonlinear Schrödinger type systems," *J. Comput. Phys.*, vol. 147, no. 1, pp. 166–186, 1998.

[12] M. I. Yousefi and F. R. Kschischang, "Information transmission using the nonlinear Fourier transform, Part II: Numerical methods," *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 4329–4345, Jul. 2014.

[13] V. Vaibhav, "Higher order convergent fast nonlinear Fourier transform," *IEEE Photon. Technol. Lett.*, vol. 30, no. 8, pp. 700–703, Apr. 15, 2018. doi: 10.1109/LPT.2018.2812808.

[14] G. Boffetta and A. R. Osborne, "Computation of the direct scattering transform for the nonlinear Schrödinger equation," *J. Comput. Phys.*, vol. 102, no. 2, pp. 252–264, Oct. 1992.

[15] S. Chimmalgi, P. J. Prins, and S. Wahls, "Nonlinear Fourier transform algorithm using a higher order exponential integrator," in *Proc. Adv. Photon.*, 2018, pp. 1–2, Paper SpM4G.5. doi: 10.1364/SPPCOM.2018.SpM4G.5.

[16] M. Thalhammer, "A Fourth-order commutator-free exponential integrator for nonautonomous differential equations," *SIAM J. Numer. Anal.*, vol. 44, no. 2, pp. 851–864, 2006. doi: 10.1137/05063042.

[17] G. P. Agrawal, *Fiber-Optic Communication Systems*, 4th ed. Hoboken, NJ, USA: Wiley, 2013.

[18] A. Shabat and V. Zakharov, "Exact theory of two-dimensional self-focusing and one-dimensional self-modulation of waves in nonlinear media," *Soviet Phys. JETP*, vol. 34, no. 1, p. 62, 1972.

[19] A. Bruckstein and T. Kailath, "An inverse scattering framework for several problems in signal processing," *IEEE ASSP Mag.*, vol. 4, no. 1, pp. 6–20, Jan. 1987. doi: 10.1109/MASSP.1987.1165567.

[20] L. D. Faddeev and L. Takhtajan, *Hamiltonian Methods in the Theory of Solitons*. Berlin, Germany: Springer-Verlag, 2007.

[21] P. J. Prins and S. Wahls, "Higher order exponential splittings for the fast non-linear Fourier transform of the Korteweg-de Vries equation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Apr. 2018, pp. 4524–4528.

[22] A. Vasylchenkova, J. Prilepsky, D. Shepelsky, and A. Chattopadhyay, "Direct nonlinear Fourier transform algorithms for the computation of solitonic spectra in focusing nonlinear Schrödinger equation," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 68, pp. 347–371, Mar. 2019. doi: 10.1016/j.cnsns.2018.09.005.

[23] S. Blanes, F. Casas, J. Oteo, and J. Ros, "The magnus expansion and some of its applications," *Phys. Rep.*, vol. 470, no. 5, pp. 151–238, 2009. doi: 10.1016/j.physrep.2008.11.001.

[24] S. Blanes, F. Casas, and M. Thalhammer, "High-order commutator-free quasi-Magnus exponential integrators for non-autonomous linear evolution equations," *Comput. Phys. Commun.*, vol. 220, pp. 243–262, Nov. 2017. doi: 10.1016/j.cpc.2017.07.016.

[25] M. Yamada and K. Sakuda, "Analysis of almost-periodic distributed feedback slab waveguides via a fundamental matrix approach," *Appl. Opt.*, vol. 26, no. 16, pp. 3474–3478, 1987. doi: 10.1364/AO.26.003474.

[26] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and Systems: Pearson New International Edition*, 2nd ed. Harlow, U.K.: Pearson, 2013.

[27] M. I. Yousefi and F. R. Kschischang, "Information transmission using the nonlinear Fourier transform, Part I: Mathematical tools," *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 4312–4328, Jul. 2014. doi: 10.1109/TIT.2014.2321143.

[28] J. Satsuma and N. Yajima, "B. Initial value problems of one-dimensional self-modulation of nonlinear waves in dispersive media," *Progr. Theor. Phys. Suppl.*, vol. 55, pp. 284–306, Jan. 1974.

[29] D. Rourke and P. Morris, "Half solitons as solutions to the Zakharov-Shabat eigenvalue problem for rational reflection coefficient with application in the design of selective pulses in nuclear magnetic resonance," *Phys. Rev. A, Gen. Phys.*, vol. 46, no. 7, pp. 3631–3636, 1992.

[30] E. V. Podivilov, D. A. Shapiro, and D. A. Trubitsyn, "Exactly solvable profiles of quasi-rectangular Bragg filter with dispersion compensation," *J. Opt. A, Pure Appl. Opt.*, vol. 8, no. 9, p. 788, 2006.

[31] D. S. Bernstein and W. So, "Some explicit formulas for the matrix exponential," *IEEE Trans. Autom. Control*, vol. 38, no. 8, pp. 1228–1232, Aug. 1993.

[32] E. H.-L. Liu, "Fundamental methods of numerical extrapolation with applications," MIT Open Courseware, Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep., 2006. Accessed: Nov. 1, 2018. [Online]. Available: http://web.mit.edu/ehliu/Public/Spring2006/18.304/extrapolation.pdf

[33] J. L. Blue and H. K. Gummel, "Rational approximations to matrix exponential for systems of stiff differential equations," *J. Comput. Phys.*, vol. 5, no. 1, pp. 70–83, 1970. doi: 10.1016/0021-9991(70)90053-7.

[34] J. Skaar and O. H. Waagaard, "Design and characterization of finite-length fiber gratings," *IEEE J. Quantum Electron.*, vol. 39, no. 10, pp. 1238–1245, Oct. 2003. doi: 10.1109/JQE.2003.817581.

[35] S. Wahls, S. Chimmalgi, and P. J. Prins, "FNFT: A software library for computing nonlinear Fourier transforms," *J. Open Source Softw.*, vol. 3, no. 23, p. 597, Mar. 2018. doi: 10.21105/joss.00597.

[36] L. Rabiner, R. Schafer, and C. Rader, "The chirp Z-transform algorithm," *IEEE Trans. Audio Electroacoustics*, vol. 17, no. 2, pp. 86–92, Jun. 1969. doi: 10.1109/TAU.1969.1162034.

[37] J. Keiner, S. Kunis, and D. Potts, "Using NFFT 3—A software library for various nonequispaced fast Fourier transforms," *ACM Trans. Math. Softw.*, vol. 36, no. 4, pp. 19:1–19:30, Aug. 2009. doi: 10.1145/1555386.1555388.

[38] S. Wahls, "Generation of time-limited signals in the nonlinear Fourier domain via b-modulation," in *Proc. Eur. Conf. Opt. Commun. (ECOC)*, Sep. 2017, pp. 1–3. doi: 10.1109/ECOC.2017.8346231.

[39] T. Gui, G. Zhou, C. Lu, A. P. T. Lau, and S. Wahls, "Nonlinear frequency division multiplexing with b-modulation: Shifting the energy barrier," *Opt. Express*, vol. 26, no. 21, pp. 27978–27990, Oct. 2018. doi: 10.1364/OE.26.027978.

[40] S. T. Le, K. Schuh, F. Buchali, and H. Buelow, "100 Gbps b-modulated nonlinear frequency division multiplexed transmission," in *Proc. Opt. Fiber Commun. Conf. Expo. (OFC)*, Mar. 2018, pp. 1–3.

[41] A. Sidi, *Practical Extrapolation Methods: Theory and Applications*, 1st ed. New York, NY, USA: Cambridge Univ. Press, 2002.

[42] Y. Choi, J. Chun, T. Kim, and J. Bae, "The Schur algorithm applied to the one-dimensional continuous inverse scattering problem," *IEEE Trans. Signal Process.*, vol. 61, no. 13, pp. 3311–3320, Jul. 2013.

[43] I. H. Sloan, *Superconvergence*. Boston, MA, USA: Springer, 1990, pp. 35–70. doi: 10.1007/978-1-4899-2593-0_2.

[44] J. E. Huss and J. A. Pennline, "A comparison of five benchmarks," Lewis Res. Center, Cleveland, OH, USA, NASA Tech. Memorandum 88956, 1987. Accessed: Jul. 6, 2019. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19870008008.pdf

[45] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proc. IEEE*, vol. 93, no. 2, pp. 216–231, Feb. 2005. doi: 10.1109/JPROC.2004.840301.

[46] A. Vasylchenkova, J. E. Prilepsky, and S. K. Turitsyn, "Contour integrals for numerical computation of discrete eigenvalues in the Zakharov&Shabat problem," *Opt. Lett.*, vol. 43, no. 15, pp. 3690–3693, Aug. 2018. doi: 10.1364/OL.43.003690.

[47] V. Aref, S. T. Le, and H. Buelow, "Modulation over nonlinear Fourier spectrum: Continuous and discrete spectrum," *J. Lightw. Technol.*, vol. 36, no. 6, pp. 1289–1295, Mar. 15, 2018. doi: 10.1109/JLT.2018.2794475.

[48] J. McNamee and V. Y. Pan, "Efficient polynomial root-refiners: A survey and new record efficiency estimates," *Comput. Math. Appl.*, vol. 63, no. 1, pp. 239–254, 2012. doi: 10.1016/j.camwa.2011.11.015.

[49] J. Aurentz, T. Mach, R. Vandebril, and D. Watkins, "Fast and backward stable computation of roots of polynomials," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 3, pp. 942–973, 2015. doi: 10.1137/140983434.

SHRINIVAS CHIMMALGI received the B.Tech. degree in mechanical engineering from the National Institute of Technology Karnataka, India, in 2015, and the M.Sc. degree in systems and control from the TU Delft, The Netherlands, in 2017, where he is currently pursuing the Ph.D. degree with the Systems and Control Department.

His research is focused on numerical methods for nonlinear Fourier transform and its applications in fiber-optic communications.

PETER J. PRINS, photograph and biography not available at the time of publication.

SANDER WAHLS received the Diploma degree in mathematics and the Ph.D. degree *(summa cum laude)* in electrical engineering from TU Berlin, in 2007 and 2011, respectively.

He is currently an Associate Professor with the Systems and Control Department, TU Delft. Before joining TU Delft, in 2014, he spent two years as a Postdoctoral Research Fellow at the Department of Electrical Engineering, Princeton University. His research is focused on computational methods in signal processing and control, currently in the areas of integrable systems, nonlinear Fourier transforms, and their application in engineering problems.

Dr. Wahls received the 2015 Johann-Philipp-Reis Award for his work on fast nonlinear Fourier transforms and the European Research Council (ERC) Starting Grant on fast nonlinear Fourier transforms, in 2016.

● ● ●