TU Delft

Delft University of Technology

A Cascaded Structure for Generalized Graph Filters

Coutino, Mario; Leus, Geert

DOI
10.1109/TSP.2021.3099630

Publication date
2022

Document Version
Final published version

Published in
IEEE Transactions on Signal Processing

Citation (APA)
Coutino, M., & Leus, G. (2022). A Cascaded Structure for Generalized Graph Filters. *IEEE Transactions on Signal Processing*, *70*, 3499-3513. Article 9496112. https://doi.org/10.1109/TSP.2021.3099630

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.

A Cascaded Structure for Generalized Graph Filters

Mario Coutino, *Member, IEEE*, and Geert Leus, *Fellow, IEEE*

***Abstract*—One of the main challenges of graph filters is the stability of their design. While classical graph filters allow for a stable design using optimal polynomial approximation theory, generalized graph filters tend to suffer from the ill-conditioning of the involved system matrix. This issue, accentuated for increasing graph filter orders, naturally leads to very large (small) filter coefficients or error saturation, casting a shadow on the benefits of these richer graph filter structures. In addition to this, data-driven design/learning of graph filters with large filter orders, even in the case of classical graph filters, suffers from the eigenvalue spread of the input data covariance matrix and mode coupling, leading to convergence-related issues as the ones observed when identifying time-domain filters with large orders. To alleviate these conditioning and convergence problems, and to reduce the overall design complexity, in this work, we propose a cascaded implementation of generalized graph filters and an efficient algorithm for designing the graph filter coefficients in both model- and data-driven settings. Further, we establish the connections of this implementation with so-called graph convolutional neural networks and demonstrate the performance of the proposed structure in different network applications. By the proposed approach, further error reduction and better design stability are achieved.**

***Index Terms*—Cascaded filters, distributed optimization, graph filtering, graph signal processing, signal processing on graphs.**

I. Introduction

SIGNAL processing over networks is experiencing an increasing interest as traditional signal processing tasks, such as statistical inference, are beeing extended to signals with an irregular support [2]–[4], e.g., social, biological and network data. To provide a theoretical framework for understanding such data, the field of graph signal processing (GSP) [5] has been developed. It naturally incorporates the relations exhibited by the network structure through an algebraic representation of the network. This representation provides a notion of shift in the graph and, at the same time, a way to norm a Euclidean space [6]–[8].

Manuscript received October 14, 2020; revised May 23, 2021; accepted July 6, 2021. Date of publication July 26, 2021; date of current version July 14, 2022. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Vincent Gripon. This work was supported in part by the ASPIRE project (project 14926 within the STW OTP programme), and in part by the Netherlands Organization for Scientific Research. The work of Mario Coutino was supported by CONACYT. A preliminary work of this paper was presented in [1]. *(Corresponding author: Mario Coutino.)*

Mario Coutino is with the Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, 2826 CD Delft, The Netherlands, and also with Radar Technology, TNO, 2597 AK Den Haag, The Netherlands (e-mail: m.a.coutinominguez@tudelft.nl).

Geert Leus is with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2826 CD Delft, The Netherlands (e-mail: g.t.l.leus@tudelf.nl).

Digital Object Identifier 10.1109/TSP.2021.3099630

Similar to time-domain filters in traditional signal processing, graph filters (GFs) [9] have become the workhorse of GSP for solving inference problems such as interpolation/estimation [2], [10] and classification/detection [4], [11]. Unfortunately, similarly to their time-domain counterparts, higher-order graph filters may present stability issues in both their application and design, e.g., quantization of filter coefficients, ill-conditioning of system matrices, etc. To tackle these problems, several works have aimed at designing robust GFs, see, e.g., [12], [13] or to leverage polynomial approximation techniques to design graph filters in a stable fashion [14].

Although research has been carried out to obtain stable GF designs, most of these efforts have been focused on the so-called classical GFs, structures that share a one-to-one relation with time-domain filters and allow for spectrum-shaping designs. For the case of more complex GFs, capable of better approximating linear operators due to their increased degrees of freedom, such as the node-variant [15] and the constrained edge-variant GFs [16], a node-based design must be performed as they do not generally accept a spectrum-shaping design. Therefore, the design of such GFs relies on system matrices constructed with shifted versions, i.e., matrix powers, of the so-called graph shift operator (GSO), i.e., the matrix representation of the network. Due to the (possibly) large spread in the eigenvalues of the GSO, the resulting system matrices used for the design of these generalized GFs tend to have a poor numerical conditioning, especially for large filter orders. This leads to instabilities in their design and to the slow convergence of iterative methods employed for finding the respective coefficients [17]. Hence, there exists a need to develop filter design methods for these structures that are numerically stable and can cope with the convergence issues of the iterative methods involved in the design.

Besides the above issues, although classical GFs benefit from spectrum-shaping-type designs, their stable design is only possible when the GF response is known a priori. That is, this kind of design is only applicable when the shape of the (discrete) spectrum of the desired linear transform is known beforehand, i.e., model-driven design. However, in many cases, a data-driven design is desirable (or is the only option). Such situations arise when the only information available about the linear transform is given in terms of input-output data. This calls for methods that identify the underlying GF by mapping the available inputs to the respective outputs. Although for classical GFs, a straightforward deconvolution-type of approach can be devised for finding the input-output (and hence the spectrum-shaping function) mapping, see, e.g., [18] for blind graph filter identification, differently from the time-domain, this method

1053-587X © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

requires the full eigendecomposition of the GSO which, in many instances, can be prohibitive. Further, for generalized GFs, these deconvolution-type of approaches are simply not possible as these structures are not guaranteed to be diagonalizable by the eigenbasis of the GSO. Therefore, a stable and data-efficient method for identifying generalized GFs from input-output data is much needed.

To deal with similar issues, time-domain filters have been designed/identified, in the data-driven setting, by means of iterative methods such as least mean squares (LMS) or recursive least squares (RLS) [19], [20], which due to their effectiveness, have already been adapted to the graph setting, see, e.g., [21]. However, it is well known that, for instance, LMS suffers from two main problems: the eigenvalue spread of the correlation matrix of the input data, and the coupling between modes of convergence [22]. Specifically, the former effect leads to a nonuniform convergence of the different filter coefficients and the latter to nonmonotonic trajectories toward convergence. Although RLS provides a way to decouple such convergence routes by means of a matrix inversion, it is known that RLS exhibits a large sensitivity to numerical accuracy, plus, the eigenvalue spread remains a problem [23]. Unfortunately, the latter problem gets accentuated when modelling auto regressive moving average (ARMA) processes, which requires higher-order finite impulse response (FIR) filters, as it has been shown that the eigenvalue spread is a nondecreasing function of the filter length [24]. In addition to all these, learning long filters requires a small step size for both LMS or RLS, which in turn slows down the convergence and increases the number of parameter updates, i.e., times that the filter coefficients are updated. Therefore, further structure has to be imposed on the filters to counteract these issues.

In this work, our goal is two-fold. First, we aim to improve the conditioning of the system matrices involved in the design of generalized GFs [9] where classical time-domain techniques are not applicable. And second, to develop a stable and efficient updating scheme for the GF identification/learning problem in the data-driven setting. To tackle these tasks, inspired by constructions used in audio for linear prediction [25], we first introduce a cascaded implementation of generalized GFs, establishing the respective connections with so-called graph neural networks (GNNs) [26], and we then introduce an efficient algorithm for learning the coefficients of the generalized GFs that is applicable to both the model- and data-driven setting.

A. Overview and Main Contributions

Though GFs are the workhorse of GSP, the stable design/identification of GFs, specially for generalized GFs, is still far from being completely achieved. Therefore, in this work, we introduce a framework based on the cascaded implementation of GFs allowing stable and efficient filter coefficient design/learning. Our contributions broadening the state-of-the-art are the following.

- We introduce a cascaded structure of distributed GFs to mitigate the effects of large GF orders on the conditioning of the GF coefficient design problem, allowing for a better numerical stability and approximation of general linear operators.
- We analyze the error surface for the least squares design of the proposed cascaded structure and show that only saddle points are introduced in the error surface of the direct implementation. This explains why stochastic gradient descent methods are ideal for designing/identifying cascaded GF coefficients.
- Exploiting the structure of the design matrices, we propose an iterative design method for cascaded GFs for the model-driven case, i.e., known data transformation. Further, under minor modifications, we adapt the proposed method to the data-driven setting. In addition, the proposed algorithms are shown to be amenable to a large-scale implementation leveraging sparsity and deep learning frameworks such as TensorFlow [27] and Keras [28].
- Through common network applications, it is shown that the proposed cascaded implementation exhibits better numerical properties than the direct GF implementation achieving lower approximation errors while saving communication rounds in the distributed setting.

B. Outline and Notation

This paper is organized as follows. Section II discusses the required background in GSP, presents common applications of GFs and provides the context of this work. Section III introduces the proposed cascaded implementation for generalized GFs and analyzes the error surface of the corresponding least squares design problem. In addition, it illustrates the relation between cascaded GFs and graph convolutional neural networks (GCNNs). To deal with the nonconvex design of the cascaded structure, an iterative algorithm, referred to as RELAX, is proposed in Section IV for the model-driven setting. Section V leverages the proposed RELAX algorithm to introduce its data-driven variant. Further, it discusses theoretical results with respect to the learning of cascaded GFs from input-output data. Section VI showcases the benefits of the proposed implementation by numerical experiments related to common network applications. Finally, Section VII concludes the paper.

Throughout this paper, we adopt the following notation. Scalars, vectors, matrices and compound linear operators are denoted by lowercase letters (x), lowercase boldface letters ($\boldsymbol{x}$), uppercase boldface letters ($\boldsymbol{X}$), and calligraphic letters ($\mathcal{X}$), respectively. $\boldsymbol{X}^\top$ and $\boldsymbol{X}^{-1}$ are the transpose and the inverse of $\boldsymbol{X}$, respectively. The Moore-Penrose pseudoinverse of $\boldsymbol{X}$ is denoted by $\boldsymbol{X}^\dagger$. $\|\boldsymbol{X}\|$ denotes an arbitrary norm defined in the space where $\boldsymbol{X}$ is defined. κ stands for the number of degrees of freedom, i.e., the number of free parameters to identify or learn. $[K]$ denotes the set $\{1, 2, \ldots, K\}$.

II. Preliminaries

A. Graph Signal Processing

Consider a graph (possibly directed) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the set of N nodes and M edges, respectively. Further, let $\boldsymbol{W}$ and $\boldsymbol{L}$ be the weighted graph adjacency matrix

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

and an appropriate Laplacian matrix,[1] respectively. Both matrix representations of the graph are valid candidates for the so-called graph shift operator $\boldsymbol{S}$ (GSO), a linear operator that induces a *frequency notion* in the graph setting [5]. Analogous to traditional signal processing, given the decomposition $\boldsymbol{S} = \boldsymbol{U}\lambda\boldsymbol{U}^{-1}$ (assuming it exists), the *graph Fourier transform* (GFT) of the signal $\boldsymbol{x} \in \mathbb{R}^N$, supported on $\mathcal{G}$, is defined as $\hat{\boldsymbol{x}} = \boldsymbol{U}^{-1}\boldsymbol{x}$. The inverse GFT is then given by $\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{x}}$. As a result, following the GSP interpretation, the eigenvalues $\lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ of the GSO are referred to as the *graph frequencies*.

Similarly as in time domain, the notion of shift can be employed to define so-called classical GFs (C-GFs). These structures are *matrix functions* [29] of the GSO, whose finite dimensionality leads to matrix polynomials in $\boldsymbol{S}$, i.e.,

$$\mathcal{F}_{\mathrm{c}}^{K}(\boldsymbol{S}; \{\phi_k\}) := \sum_{k=1}^{K} \phi_k \boldsymbol{S}^{k-1} = \boldsymbol{U}\left[\sum_{k=1}^{K} \phi_k \lambda^{k-1}\right]\boldsymbol{U}^{-1}, \quad (1)$$

where $K \leq N$ denotes the order of the matrix polynomial. Therefore, for a given input $\boldsymbol{x} \in \mathbb{R}^N$, the output of the GF in (1), i.e., $\boldsymbol{y} = \mathcal{F}_{\mathrm{c}}(\boldsymbol{S})\boldsymbol{x}$, can be seen as the linear combination of shifted versions of the input, i.e., $\boldsymbol{x}_k = \boldsymbol{S}^k\boldsymbol{x}$. Due to the locality of $\boldsymbol{S}$, i.e., its support is defined by the connections in the graph, for physically meaningful systems, e.g., sensor networks, a shift can be implemented in a single communication round. Hence, the filtering operation can be implemented in a distributed fashion in K communication rounds.[2]

Recently, several efforts [9], [14] have been taken to increase the flexibility of the graph operation (1). Such efforts have focused on increasing the degrees of freedom of the filtering operation, while maintaining its distributed nature. In [30], the so-called *constrained edge-variant* graph filter (CEV-GF) has been introduced. The structure of the CEV-GF is given by

$$\mathcal{F}_{\mathrm{cev}}^{K}(\boldsymbol{S}; \{\boldsymbol{\Phi}_k\}) := \sum_{k=1}^{K} \boldsymbol{\Phi}_k \boldsymbol{S}^{k-1}, \quad (2)$$

where $\boldsymbol{\Phi}_k$ is a *local matrix* with the same support as $\boldsymbol{S} + \boldsymbol{I}$. From (2), it can be observed that the so-called *node-variant* graph filter (NV-GF), proposed in [15], is a particular case of the CEV-GF where the matrices $\{\boldsymbol{\Phi}_k\}_{k=1}^{K}$ are restricted to be diagonal matrices, i.e.,

$$\mathcal{F}_{\mathrm{nv}}^{K}(\boldsymbol{S}; \{\boldsymbol{\phi}_k\}) := \sum_{k=1}^{K} \mathrm{diag}(\boldsymbol{\phi}_k) \boldsymbol{S}^{k-1}. \quad (3)$$

Due to the increased degrees of freedom of these more complex structures, with respect to classical GFs [cf. (1)], they have been shown to obtain a better performance while reducing the number of communication rounds in typical network applications, see., e.g., [9]. In addition, despite that for arbitrary local matrices $\{\boldsymbol{\Phi}_k\}_{k=1}^{K}$ the filter in (2) does not directly carry the same graph frequency interpretation as (1), in [9] a restricted subfamily of (2) has been shown to have a graph frequency interpretation.

[1]Here, we do not further select a particular type of Laplacian matrix due to the different possible options, e.g., combinatorial Laplacian, in/out degree Laplacian, normalized Laplacian, etc.

[2]For these GFs, the order is in fact $K-1$ and the number of communication rounds is $K-1$, however, in view of the later generalizations, we prefer to employ K instead of $K-1$.

B. Graph Filter Applications

Graph filters are useful to describe processes running on the network structure, in which the observable variable $\boldsymbol{x} \in \mathbb{R}^N$ may be represented using *few modes* of the graph. Such processes are often referred to as *band-limited* processes [31] and have found applications in analyzing network data such as health-related data, e.g., brain [32], heart [33], or weather data and point clouds, e.g., [34]. Furthermore, the generalization of graph filters [cf. (2)] has been successfully employed for obtaining distributed approximations to arbitrary linear operators such as beamformers (for array processing) and consensus.

As graph filters provide a natural *regularization* mechanism, by including the structural information of the graph into the problem, they have found further applications in graph convolutional deep neural networks [26]. In such cases, by using a proper parametrization of the graph filters, it is possible to obtain an increased prediction performance [35] by leveraging the structure present in the data inherited by the graph topology. In addition, GFs are the building block of both the graph scattering transform [36] and personalized recommendation systems, see, e.g., [37].

C. Context

Conventional graph filter design either focuses on one-shot designs [14], [15] or on iterative designs for autoregressive moving average (ARMA) structures [38], [39]. Despite that these methods are able to cope with common filtering tasks, the resulting filter might require a large number of communication rounds to achieve a desired performance. Furthermore, if traditional time-domain designs are extended to the graph setting, the obtained filters will be restricted to be shift invariant, i.e., polynomials of the GSO, leading to inappropriate filters for approximating arbitrary linear operators by means of graph filters. Moreover, the works on robust design for GFs mostly focus on classical GFs and model-driven settings or on the theoretical understanding of effects such as quantization in the GF processing chain, see, e.g., [12], [13], [40]. Hence they either do not address the data-driven setting for (classical) generalized GFs case and do not deal with the numerically problems in the design stage due to the conditioning of the matrices.

Therefore, in this work, we focus on the problem of stable design of generalized graph filters for arbitrary linear operators. That is, we aim to provide design algorithms that *ameliorate the conditioning of the system matrices involved in the design/learning of generalized graph filters.*

III. Cascaded Implementation of Graph Filters

One of the main problems of graph filter design is the numerical stability of the least squares problems involved. Despite that for classical GFs (C-GFs) [cf. (1)] polynomial fitting in the spectral domain, by means of the Chebyshev polynomial

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

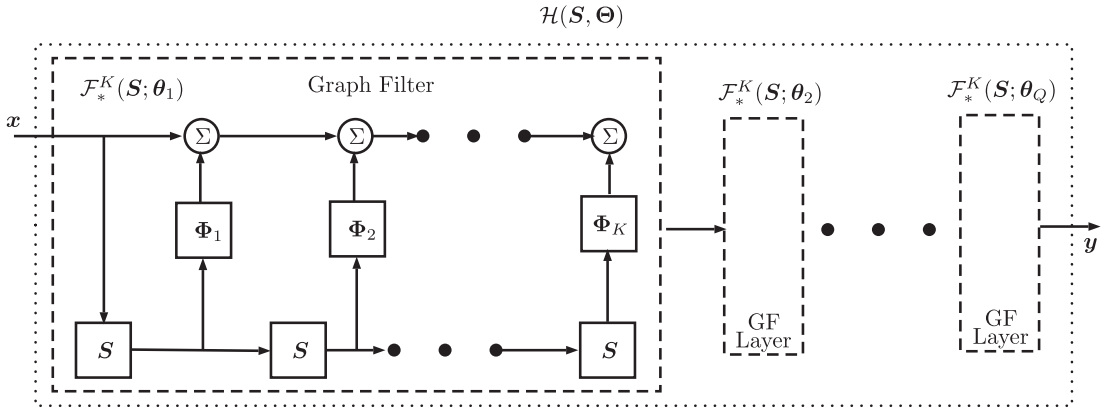


Fig. 1. Schematic of the cascaded GF implementation. Each GF can be considered as a layer of a GCN with a linear activation function. Here, we consider the most general form of GF [cf. (2)], however, the local matrices $\{\boldsymbol{\Phi}_k\}_{k=1}^K$ can be specialized to $\{\phi_k \boldsymbol{I}\}_{k=1}^K$ and $\{\text{diag}(\boldsymbol{\phi}_k)\}_{k=1}^K$ for C-GFs and NV-GFs, respectively.

expansion [41], can be employed for large polynomial orders, other types of graph filters rely on a *node-domain* design, i.e., entry-wise fitting with respect to the target linear operator, $\boldsymbol{H}^*$; that is,

$$\min_{\boldsymbol{\theta}} \|\mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}) - \boldsymbol{H}^*\|_{\text{F}}, \tag{4}$$

where $\mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta})$ is a GF of order K with parameters $\boldsymbol{\theta} \in \mathbb{R}^{K\kappa}$; in this notation, the asterisk is a space holder for the particular kind of GF employed, e.g., $\mathcal{F}_{\text{c}}^K$, $\mathcal{F}_{\text{nv}}^K$ or $\mathcal{F}_{\text{cev}}^K$; and κ denotes the number of degrees of freedom of the particular kind of GF, e.g. $\kappa = 1$ for the C-GF, $\kappa = N$ for the NV-GF and $\kappa = M$ for the CEV-GF.

Unfortunately, the LS problem in (4) quickly becomes ill-conditioned when the order increases. This issue leads to slow convergence of iterative methods for solving the least squares problem, memory issues due to low sparsity levels of the system matrices, or/and unstable solutions with very large (very small) filter coefficients. To deal with the conditioning issues of the system matrices in the node-domain design, and borrowing ideas from traditional cascaded implementations of finite- and infinite-impulse response filters [23], [42], we put forth a cascaded implementation of graph filters. That is, we propose to limit the maximum order of a given GF and employ this GF module as a building block of a larger system.

Mathematically, we introduce the graph operation as

$$\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta}) := \prod_{i=1}^{Q} \mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}_i) = \mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}_Q)\cdots\mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}_1), \tag{5}$$

where $\boldsymbol{\Theta} = [\boldsymbol{\theta}_1^T,\ldots,\boldsymbol{\theta}_Q^T]^T \in \mathbb{R}^{QK\kappa}$. Notice that when the GF is a C-GF, (5) is another C-GF of order $\max\{KQ, N\}$. This is however not the case for both the NV-GF and CEV-GF unless the coefficient matrices have a particular structure, i.e., they enforce shift-invariance. The implementation of the proposed construction is shown in Fig. 1.

The adopted cascaded implementation is similar to the cascading of biquad filters [43] in time-domain processing for attenuating the effect of quantizing the filter coefficients. They are also similar to the structures employed for adaptive linear predictors [23] which are robust against conditioning of the input covariance matrix and accept larger step sizes in their updating steps as the coefficient sensitivities of a cascaded filter are much lower than that of the direct form [44]. Thus, similar to the motivations present in the time domain, by introducing such a filter implementation, we aim to: (i) improve the conditioning of the design problem, (ii) reduce the complexity of the optimization problems involved and (iii) achieve a better performance with a reduced order. However, all these advantages do not come for free: we need to give up the *convexity* of the overall filter design problem, i.e.,

$$\arg\min_{\{\boldsymbol{\theta}_i\}_{i=1}^Q} \|\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta}) - \boldsymbol{H}^*\|_{\text{F}}. \tag{6}$$

As the design of the direct implementation, i.e., a single GF with large order, is a convex problem [cf. (4)], it is theoretically guaranteed that an optimal configuration of coefficients can be efficiently found. However, due to the nonconvexity of the design problem for the cascaded problem [cf. (6)], this property cannot be guaranteed without understanding the error surface for this kind of problem. In the following, we present an analysis of the error surface for a particular family of problems of the form (6).

A. Theoretical Study of Cascaded GF Error Surface

Surprisingly, similar to the results in system identification and adaptive filtering [44], it is possible to show that the error surface of the cascaded implementation of a particular family of GFs exhibits critical points that are either global minimizers or saddle points. To show this, we first introduce some basic notions that characterize what happens to an error surface when a new parametrization is introduced. And then, we use these results to show the type of critical points that the error surface of the cascaded implementation of a particular family of GFs has.

Critical points after reparametrization: Let us consider two *equivalent* implementations of a GF of order K, one in direct form with parameters $\boldsymbol{\theta}_{\text{d}} = [\theta_1,\ldots,\theta_{K_{\text{d}}}]^\top$ and the other in cascaded form with parameters $\boldsymbol{\Theta}_{\text{c}} = [\boldsymbol{\theta}_{\text{d}_1}^\top,\ldots,\boldsymbol{\theta}_{\text{d}_Q}^\top]$, where

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

$\boldsymbol{\theta}_{\mathrm{d}_i} \in \mathbb{R}^{K_\mathrm{c}}\ \forall\ i$ are the parameters of the ith cascaded module; that is,

$$\mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}_\mathrm{d}) = \mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta}_\mathrm{c}), \tag{7}$$

where the LHS denotes the direct implementation and the RHS the cascaded implementation. Further, let us denote with $\mathcal{D}_\mathrm{d}$ and $\mathcal{D}_\mathrm{c}$ the sets of feasible direct and cascaded GF coefficients of given order and kind, e.g., C-GF, NV-GF, etc., respectively. That is, if C-GFs of order K are selected, $\mathcal{D}_\mathrm{d}$ contains all the C-GFs of order at most K. Similarly, the equivalent set $\mathcal{D}_\mathrm{c}$ contains all the cascaded C-GFs with order at most K formed by modules of order K_c. Hence, $\boldsymbol{\theta}_\mathrm{d} \in \mathcal{D}_\mathrm{d}$ and $\boldsymbol{\Theta}_\mathrm{c} \in \mathcal{D}_\mathrm{c}$.

The above implies that the LS error with respect to a desired response $\boldsymbol{H}^*$ is the same for both implementations (realizations). However, while for the case of the direct form the LS error function is convex with respect to the GF coefficients [cf. (4)], for the cascaded implementation, the LS error is clearly nonconvex [cf. (6)]. So, despite that the error surface for the direct implementation, due to its convexity, has only a global minimum, the cascaded implementation could have several local minima, changing the modality of the LS error surface.

To characterize the change in the modality of the cost function, we introduce the following adapted Lemma from [44].

Lemma 1: Let a mapping $\varphi(\cdot) : \mathcal{D}_\mathrm{d} \to \mathcal{D}_\mathrm{c}$ exist and be continuous and surjective (onto), then all the newly formed stationary points (critical points) are saddle points.

Proof: See Appendix A.[3] ■

The previous Lemma guarantees that the original critical points are maintained and that all new critical points introduced are saddle points if a proper reparametrization is used. Here, proper refers to the properties that the mapping φ must exhibit. Unfortunately, as shown next, these properties cannot be guaranteed for all types of GFs. This fact is made formal in the next result.

Theorem 1: The continuous and surjective map φ exists for cascaded GFs whose modules are first-order C-GFs.

Proof: See Appendix B. ■

Corollary 1: The map φ also exists if instead of C-GF modules, ARMA C-GF modules, see, e.g., [38], are used to implement the cascaded GF.

Saddle points of Cascaded C-GFs: Due to Theorem 1, we know that only C-GFs allow for continuous and surjective mappings, thus due to Lemma 1 we can ensure that all the newly created critical points are saddle points, hence no local/global minima. The following result identifies where these saddle points, due to the cascaded reparametrization, appear in the parameter space.

Theorem 2: Consider a direct implementation of a C-GF

$$\boldsymbol{H} = \sum_{k=0}^{K} \phi_k \boldsymbol{S}^K, \text{ with } \phi_0 = 1,$$

and a cascaded GF, $\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta})$, of the form (5) with $Q = K$ modules. Further, let the modules be given by

$$\mathcal{F}_\mathrm{c}^1(\boldsymbol{S};\boldsymbol{\theta}_q) = \boldsymbol{I} + \phi_1^{(q)}\boldsymbol{S}, \forall\ q \in [Q].$$

Then, the newly introduced saddle points in (6), with respect to a desired response $\boldsymbol{H}^*$, are found in a manifold where any two or more graph filter coefficients are the same.

Proof: See Appendix C. ■

Corollary 2: The above result also holds in the case that the modules are ARMA_1 C-GFs, i.e.,

$$\mathcal{F}_\mathrm{c}^1(\boldsymbol{S};\boldsymbol{\theta}_q) = (\boldsymbol{I} + \psi_1^{(q)}\boldsymbol{S})^{-1}(\boldsymbol{I} + \phi_1^{(q)}\boldsymbol{S}), \forall\ q \in [Q]$$

where $\boldsymbol{\theta}_q = [\psi_1^{(q)}, \phi_1^{(q)}]^\top$ are the ARMA_1 C-GFs parameters.

The result of Theorem 2 shows that saddle points only appear when two modules (stages) of C-GFs have the same coefficients. This is in line with the results of system identification stating that the modality of the error surface for cascaded IIR filters are affected by multiple poles or zeros [45]. In addition, although the result of Theorem 2 seems restrictive, i.e., it is stated for C-GFs of order one, the theorem can be extended to second- (or higher-) order modules. However, for the sake of exposition, we here consider first-order modules as they are the simplest GF units and they are the basis for graph neural networks (GNNs) and residual GNNs (RGNNs), see, e.g., [46]. Further, the restriction to $\phi_0 = 1$ is w.l.o.g. as for any other value different from unity, the same result can be obtained by proper scaling.

To conclude this section, we bring to attention the following. First, the fact that only saddle points are introduced when a C-GF is reparametrized as a cascade of C-GFs [cf. Theorem 2] indicates that *stochastic* descent methods, e.g., stochastic gradient descent (SGD) or LMS, are good candidates for minimizing the cost in (6). The stochastic nature of such algorithms provides a natural protection against saddle points. This is because the jitter present in these methods allows them to escape from saddle points. This key observation is the working assumption for the optimization methods used in state-of-the-art machine learning methods for optimizing nonconvex costs, see, e.g., [47]. Second, the negative result obtained for other kinds of GFs [cf. Theorem 1] suggests that although traditional descent methods might work "sufficiently well," we should put efforts on devising methods that are not only *globally convergent* but that are *efficient*. This last aspect motivates the algorithm development presented in Section IV. There we propose an iterative method, amenable for sparse iterative solvers and deep learning optimization algorithms, to fit/learn the filter coefficients of (5).

Before introducing the proposed coefficient learning algorithm, in the following part, we present the relation between the proposed cascaded GF implementation and graph convolutional networks.

B. Relation to Graph Convolutional Networks

In recent years, machine learning over graphs has drawn an increasing amount of attention [48]–[50]. However, despite that the graph structure provides a highly informative prior, the task of learning over graphs still remains highly complex due to this same structure. As a result, *graph convolutional neural networks* (GCNNs), which leverage the structural information of the graph, have been put forth [26]. GSP and more specifically graph filtering provide a formal understanding of the basic operations involved in GCNNs [35].

[3]The appendix is found in the supplemental material (extended version).

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

Formally, a GCNN is a neural network that operates on graph data and whose *hidden layers* can be represented as

$$\boldsymbol{X}^{(i)} = \rho(\boldsymbol{S}; \boldsymbol{X}^{(i-1)}), \tag{8}$$

where $\boldsymbol{X}^{(0)}$ is the original $N \times F^{(0)}$ input data (data matrix $\boldsymbol{X}$), $\boldsymbol{S}$ the GSO and ρ a propagation rule [51]. Thus, the hidden layer $\boldsymbol{X}^{(i)}$ can be interpreted as an $N \times F^{(i)}$ feature matrix whose ith row represents the features of the ith node. Under this setting, by stacking Q different layers, i.e.,

$$\mathcal{H}_{\text{gcnn}}(\boldsymbol{S}; \boldsymbol{X}^{(0)}) := \rho(\boldsymbol{S}; \rho(\boldsymbol{S}; \rho(\boldsymbol{S}; \dots \rho(\boldsymbol{S}; \boldsymbol{X}^{(0)}) \dots))), \tag{9}$$

we obtain a so-called GCNN. Note that different flavors of GCNNs can be obtained by choosing different alternatives for the propagation rule. Typically, the propagation rule is selected as

$$\rho(\boldsymbol{S}; \boldsymbol{X}^{(i)}) := \sigma(\boldsymbol{S}\boldsymbol{X}^{(i)}\boldsymbol{W}^{(i)}), \tag{10}$$

where $\sigma(\cdot)$ is an element-wise non-linearity, e.g., a linear rectifier unit (ReLU), and $\boldsymbol{W}^{(i)}$ a weight matrix to combine the features of the nodes. From (10), we observe that the number of features at the $(i+1)$th layer is defined by the number of columns of $\boldsymbol{W}^{(i)}$ and hence $\boldsymbol{W}^{(i)}$ is an $F^{(i)} \times F^{(i+1)}$ matrix.

Observing the expression involved in a GCNN, we can easily draw connections between the model (5) and a GCNN. For the filtering task, we are mostly concerned with one-dimensional signals at each node, hence the number of features at *all layers* can remain constant and equal to one. Thus, the weight matrices $\{\boldsymbol{W}^{(i)}\}$ are reduced to simple scalars. Further, despite that a single shift with respect to the GSO captures certain structural properties, a GF is able to highlight different (and possibly more complex) properties of the graph structure in a parsimonious manner, i.e., through a parametrized representation. Further, considering that we want to build a linear model-driven system, i.e., in most GF applications a desired linear operator is given to be approximated, the element-wise non-linearity $\sigma(\cdot)$ is not required and an identity function suffices, i.e., $\sigma(x) = x$. As a result, by using these considerations, we can see that (9) reduces to (5), where each layer is a first-order C-GF and the input is the graph signal $\boldsymbol{x}$. Finally, we note that other architecture that can be seen as particular instance of the cascaded implementation (if a non linearity is applied to its output) is the simple graph convolution (SGC) network [52]. In this light, a SGC of order K can be seen a single GF module, $Q = 1$ where $\theta_i = 0 \; \forall \; 1 \leq i < K$ and $\theta_K = 1$.

IV. RELIEF Algorithm

Let us consider again the model for the cascaded GF implementation

$$\mathcal{H}(\boldsymbol{S}; \boldsymbol{\Theta}) = \prod_{i=1}^{Q} \mathcal{F}_*^K(\boldsymbol{S}; \boldsymbol{\theta}_i), \tag{11}$$

where the parameter dependency has been stated explicitly, and $\boldsymbol{\Theta} := [\boldsymbol{\theta}_1^T, \dots, \boldsymbol{\theta}_Q^T]^T \in \mathbb{R}^{Q\kappa}$ is the parameter vector of the cascaded GF implementation.

The *general* filter design task consists of the following optimization problem: given a linear operator $\boldsymbol{H}^* \in \mathbb{R}^{N \times N}$ find

$$\boldsymbol{\Theta}^* = \underset{\{\boldsymbol{\theta}_i \in \mathcal{C}\}_{i=1}^{Q}}{\arg\min} \; \left\| \mathcal{H}(\boldsymbol{S}; \boldsymbol{\Theta}) - \boldsymbol{H}^* \right\|, \tag{12}$$

where $\mathcal{C}$ is a desired feasible set, e.g., interval, normed ball, etc., and $\|\cdot\|$ a desired norm, i.e., spectral normal, Frobenius norm. Notice that because the operator to approximate is linear, and that there are no data-dependent non-linearities in $\mathcal{H}$, *input/output data* is not required for solving (12). This type of design is what we refer to as *model-based design*.

As discussed before, one of the main challenges for finding the parameter vector $\boldsymbol{\Theta}^*$, even for a simple (or convex) feasible set $\mathcal{C}$, is that its components interact in the cost function of (12) in a non-convex manner. Therefore, the cost function is generally a multi-modal function with several local minima and off-the-shelf convex solvers can not be employed directly.

An alternative to deal with the non-convexity of the cost function in (12) but still being able to use readily available efficient convex solvers, is to perform *sequential fitting* for each of the GF modules. That is, given a certain tolerance, i.e., error between the fitted GF and desired linear operator, we start by solving

$$\boldsymbol{\theta}_1^* = \underset{\boldsymbol{\theta}_1 \in \mathcal{C}}{\arg\min} \; \left\| \mathcal{F}_*^K(\boldsymbol{S}; \boldsymbol{\theta}_1) - \boldsymbol{H}^* \right\|, \tag{13}$$

and proceed with the remaining filters by sequentially solving

$$\boldsymbol{\theta}_q^* = \underset{\boldsymbol{\theta}_q \in \mathcal{C}}{\arg\min} \; \left\| \mathcal{F}_*^K(\boldsymbol{S}; \boldsymbol{\theta}_q) \left[\prod_{i=1}^{q-1} \mathcal{F}_*^K(\boldsymbol{S}; \boldsymbol{\theta}_i^*) \right] - \boldsymbol{H}^* \right\|, \tag{14}$$

This way, a solution $\hat{\boldsymbol{\Theta}}$ within the desired tolerance can be obtained.

Although this is a straightforward approach, it has several drawbacks. For instance, in this approach each filter is fitted once, hence if early stages result in a bad fit this cannot be corrected for in later stages, thus the required (desired) tolerance might not be achievable.

To alleviate this issue, we propose to fit the filters using ideas similar to the ones employed in the RELAX method for mixed-spectrum estimation [53], where in every stage, after a set of parameters has been estimated, the old set of parameters are refitted to improve the cost function value. However, instead of refitting all GFs every time a new GF is added, as in the case of RELAX, we restrict ourselves to the left and right most filters. Hence, the name of the method: right (REchts) -left (LInks) itErative Fitting (RELIEF). In addition, to avoid the explicit computation of the inverse filters, we use a sparse construction (which allows a matrix free implementation) for solving the LS design.

In the following, we first introduce the sparse construction for fitting individual GFs (`linSparseSolve` routine) and then the refitting routine to improve the cost function value after each stage of the algorithm (`refitPair` routine). A summary of the proposed method is shown in Algorithm 1.

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

A. *linSparseSolve Routine*

Without loss of generality, let us focus on the cascaded GF implementation of order q given by

$$\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta}) = \boldsymbol{H}_{\rm l}\boldsymbol{M}\boldsymbol{H}_{\rm r}, \tag{15}$$

where $\boldsymbol{H}_{\rm r} := \mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}_1)$, $\boldsymbol{H}_{\rm l} := \mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}_q)$ and

$$\boldsymbol{M} := \prod_{i=2}^{q-1} \mathcal{F}_*^K(\boldsymbol{S};\boldsymbol{\theta}_i). \tag{16}$$

The minimizer of problem (14) for $\boldsymbol{H}_{\rm l}$ with $\boldsymbol{M}$ and $\boldsymbol{H}_{\rm r}$ fixed, assuming $\mathcal{C} = \mathbb{R}^N$, can be shown to be given by the solution of the least squares problem

$$\arg\min_{\boldsymbol{\theta}_q} \|\boldsymbol{\Omega}_{\rm l}\boldsymbol{\theta}_q - \mathrm{vec}(\boldsymbol{H}^*)\|_2, \tag{17}$$

where $\boldsymbol{\Omega}_{\rm l} := (\boldsymbol{H}_{\rm r}^T\boldsymbol{M}^T \otimes \boldsymbol{I})\boldsymbol{\Psi}$ with $\otimes$ the Kronecker product, $\mathrm{vec}(\cdot)$ is the vectorization operation and

$$\boldsymbol{\Psi} := [(\boldsymbol{I}\otimes\boldsymbol{I})\boldsymbol{J}, (\boldsymbol{S}^T\otimes\boldsymbol{I})\boldsymbol{J}, \ldots, ((\boldsymbol{S}^T)^K\otimes\boldsymbol{I})\boldsymbol{J}] \in \mathbb{R}^{N^2\times K\kappa}, \tag{18}$$

is the GF system matrix, with $\boldsymbol{J}$ an $N^2\times\kappa$ selection matrix that only preserves the nonzero entries of $\mathrm{vec}(\boldsymbol{\Phi}_k)$. Notice that by construction, the Kronecker matrix is a *sparse matrix* and for low filter orders, i.e., below the diameter of the graph, the $\boldsymbol{\Psi}$ matrix is also sparse. Hence, it can be stored efficiently in memory if it is desired to construct $\boldsymbol{\Omega}_{\rm l}$ explicitly. Least squares problems as (17) can be easily solved by methods such as LSMR [54]. In addition, if $\boldsymbol{\Omega}_{\rm l}$ is explicitly computed, we can build a preconditioner to accelerate the convergence of LSMR by scaling the columns of $\boldsymbol{\Omega}_{\rm l}$ such that every column has unit 2-norm. For very large systems, we may prefer to keep $\boldsymbol{\Omega}_{\rm l}$ as an *operator*. That is, we avoid its construction explicitly and only provide a routine which computes the actions $\boldsymbol{\Omega}_{\rm l}\boldsymbol{x}$ and $\boldsymbol{\Omega}_{\rm l}^T\boldsymbol{x}$ for an arbitrary vector $\boldsymbol{x}$. Algorithms as LSMR can still be employed in this setup, however, the preconditioning of the system is not simple unless the norm of the columns of $\boldsymbol{\Omega}_{\rm l}$ can be estimated accurately. Note that, as we assume that (17) is not necessarily consistent, i.e., large relative tolerance, LSMR is preferred over the popular LSQR method [55] due to its faster convergence. In Algorithm 1, the routine `linSparseSolve`$(\boldsymbol{A},\boldsymbol{b})$ makes reference to the procedure of solving a least squares problem $\boldsymbol{A}\boldsymbol{x}\approx\boldsymbol{b}$ such as (17) by means of LSMR. For completeness, the LSMR algorithm, specialized for solving problems of the form (17), is provided in Appendix D.

A similar system can be obtained for fitting $\boldsymbol{H}_{\rm r}$ for a fixed $\boldsymbol{H}_{\rm l}$ and $\boldsymbol{M}$ by making minor changes. Hence, for sake of space, we omit this derivation.

B. *Refitpair Routine*

Similar to RELAX, we would like to *correct* for a possibly wrong fitting, without going to the extreme of doing a multi-block block-coordinate-descent (BCD) [56], where only a set of GFs are fitted at a time, while keeping the rest constant and then iterating until convergence (if achieved). Although such an approach is possible, it has some drawbacks. First, to the best of our knowledge, for multi-block BCD there are no guarantees on the behaviour of limit points of the generated sequences in the general case. Hence, to guarantee convergence, i.e., each subproblem is convex [56], each block has to be updated individually. Second, even if individual GF updates are considered, the fitting of that many GFs and the possibly slow convergence renders the approach unattractive. And third, if a direct solver with an explicit system matrix is used, the sparsity of the system matrix is lost.

Therefore, instead of iterating over all filters, at every step, we propose to refit the left- and right-most filters [cf. (15)]. This not only allows to have convergence guarantees, i.e., two-block BCD convergence is proved in [57], but also to have a reduced number of subproblems, which (i) effectively reduces the complexity of the overall approach and (ii) makes use of the sparsity of the involved system matrices. A summary of this routine is shown in Algorithm 2.

Algorithm 1: `RELIEF` Algorithm.

Result: $\{\boldsymbol{\theta}_i\}_{i\in[Q]}$: filter parameters
Input: $\boldsymbol{H}^*$, $\boldsymbol{\Psi}$, Q, maxIt, $\epsilon_{\rm tol}$
initialization: $\boldsymbol{\theta}_i = \boldsymbol{0}\,\forall\, i\in[Q]$, $q=0$, $\boldsymbol{M} = \boldsymbol{H}_1 = \boldsymbol{I}$, $\boldsymbol{h}^* = \mathrm{vec}(\boldsymbol{H}^*)$;
while $(\epsilon > \epsilon_{\rm tol})$ & $(q < Q)$ **do**
  $q = q+1$;
  $\boldsymbol{H}_q \leftarrow$ `linSparseSolve`$([\boldsymbol{H}_1^T\boldsymbol{M}^T\otimes\boldsymbol{I}]\boldsymbol{\Psi}, \boldsymbol{h}^*)$;
  if $q > 1$ **then**
    $(\boldsymbol{H}_1, \boldsymbol{H}_q) \leftarrow$ `refitPair`$(\boldsymbol{H}^*, \boldsymbol{\Psi}, \boldsymbol{H}_q, \boldsymbol{H}_1, \boldsymbol{M}, \mathrm{maxIt}, \epsilon_{\rm tol})$;
    $\boldsymbol{M} \leftarrow \boldsymbol{H}_q\boldsymbol{M}$
  end
  $\boldsymbol{H}_{\rm total} \leftarrow \boldsymbol{M}\boldsymbol{H}_1$;
  $\epsilon \leftarrow \|\boldsymbol{H}_{\rm total} - \boldsymbol{H}^*\|_{\rm F}^2$;
end

Algorithm 2: `refitPair` Routine.

Result: $(\boldsymbol{\theta}_{\rm l}, \boldsymbol{\theta}_{\rm r})$: filter parameters
Input: $\boldsymbol{H}^*$, $\boldsymbol{\Psi}$, $\boldsymbol{H}_{\rm l}$, $\boldsymbol{H}_{\rm r}$, $\boldsymbol{M}$, maxIt, $\epsilon_{\rm tol}$
initialization: $\mathrm{numIt} = 0$, $\boldsymbol{h}^* = \mathrm{vec}(\boldsymbol{H}^*)$;
while $(\epsilon > \epsilon_{\rm tol})$ & $(\mathrm{numIt} < \mathrm{maxIt})$ **do**
  $\mathrm{numIt} = \mathrm{numIt} + 1$;
  $\boldsymbol{H}_{\rm r} \leftarrow$ `linSparseSolve`$([\boldsymbol{I}\otimes\boldsymbol{H}_{\rm l}\boldsymbol{M}]\boldsymbol{\Psi}, \boldsymbol{h}^*)$;
  $\boldsymbol{H}_{\rm l} \leftarrow$ `linSparseSolve`$([\boldsymbol{H}_{\rm r}^T\boldsymbol{M}^T\otimes\boldsymbol{I}]\boldsymbol{\Psi}, \boldsymbol{h}^*)$;
  $\epsilon \leftarrow \|\boldsymbol{H}_{\rm l}\boldsymbol{M}\boldsymbol{H}_{\rm r} - \boldsymbol{H}^*\|_{\rm F}^2$;
end

V. Data-Driven RELIEF

Though model-driven designs are appealing because they do not require input/output data, in many cases, the only information available of the underlying transform is given through input-output pairs, $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^P$. Examples of such scenarios appear in system network identification, e.g., [58], [59], or graph identification tasks, see, e.g., [18], [60], [61]. Therefore, there

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

is a need to put forth a *data-driven method* for cascaded GF coefficient *learning*.

Formally, a data-driven design for the cascaded GF in (5) and in the unconstrained case, solves the following optimization problem

$$\boldsymbol{\Theta}^* = \underset{\{\boldsymbol{\theta}_i\}_{i=1}^Q}{\arg\min} \sum_{i=1}^{P} \|\boldsymbol{y}_i - \mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta})\boldsymbol{x}_i\|_2^2 \tag{19}$$

Despite that (19) can be solved by deterministic variants of descent methods, or by substituting the cascaded GF by a higher-order GF, there are several issues that make these approaches not so attractive. First, it is well-known from the time-domain literature, see, e.g., [23], [44], that the sensitivities of coefficients of cascaded structures are much lower than those of direct forms; that is, the deviation of the magnitude response due to a small change in the coefficients of cascaded forms is much smaller than for direct forms. Second, deterministic approaches to solve (19) might lead to problems of high memory and computational cost as for large P, i.e., a large number of data pairs, computing the full gradient might be prohibitive. Plus, batch-processing benefits, e.g., parallelization, are not leveraged.

In addition to above issues, even a direct solution, i.e.,

$$\boldsymbol{\Theta}^* = \underset{\{\boldsymbol{\theta}_i\}_{i=1}^Q}{\arg\min} \|\boldsymbol{Y}\boldsymbol{X}^\dagger - \mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta})\|_{\mathrm{F}}^2 \tag{20}$$

where $\boldsymbol{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_P]$ and $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P]$, is not desirable due to the initial effort of finding the pseudoinverse of the input data, i.e., $\boldsymbol{X}^\dagger$.

In addition, due to the result of Theorem 2, when the cascaded implementation and a stochastic (batch) variant of gradient descent is employed, such as LMS or SGD, there is no risk to be stuck in a saddle point due to the jitter present in the method. Hence, in the following, we introduce a batch version of RELIEF for the data-driven setting.

A. Staggered RELIEF

To adapt RELIEF to the data-driven setting, let us consider the original three-way factorization of the cascaded GF used by RELIEF in the matrix version of the cost in (19), i.e.,

$$\|\boldsymbol{Y} - \boldsymbol{H}_{\mathrm{l}}\boldsymbol{M}\boldsymbol{H}_{\mathrm{r}}\boldsymbol{X}\|_{\mathrm{F}}^2. \tag{21}$$

From (21), we observe that now there is a matrix to the right of $\boldsymbol{H}_{\mathrm{r}}$ which means that the sparsity present in the system matrix exploited in RELIEF is lost. Hence, some computational benefits of the approach are lost. In addition, as discussed in (20), although directly taking the pseudoinverse of the input data matrix would allow for a direct application of RELIEF, it requires a costly inversion. Therefore, we put forth a small variant of RELIEF to cope with these details and allow for batch updates.

To tackle the problems with a vanilla implementation of RELIEF in the data-driven setting, we propose the following updating scheme for the left and right GF coefficients. First, we replace the routine `linSparseSolve` in Algorithm 1 by the routine `descentMethod`. Here, the routine `descentMethod` is any stochastic variant of a descent method, e.g., SGD, RMSprop [62], Adam [63], etc., working under mini-batch assumptions and allowing a scheduler for step-size and batch-size. The latter requirement is to leverage the benefits of parallelization and a reduction of coefficient updates due to proper batch size scheduling, see, e.g., [64]. Second, the routine `refitPair` is substituted by a process summarized in Fig. 2. In this process, we refit and update, in two stages, the left and right coefficients, similar to the model-driven RELIEF. As seen in Fig. 2, both the left and right updates have a feedback loop of the error. Hence, they are also implemented as descent methods. However, due to the linearity of the operations, i.e., the transform is data independent, we can collapse, at every iteration, the right chain into a single transformation; that is, we define

$$\tilde{\boldsymbol{H}}_{\mathrm{r}} = \boldsymbol{M}\boldsymbol{H}_{\mathrm{r}}, \tag{22}$$

and process (possibly in parallel) the input to obtain the "filtered" version $\tilde{\boldsymbol{X}} := \tilde{\boldsymbol{H}}_{\mathrm{r}}\boldsymbol{X}$. Though this step might be seen as trivial, it avoids multiplication-induced overheads that might appear if each GF is defined as a *layer* in a deep learning framework e.g., as in Tensorflow. This filtered version can then be used as input to the `descentMethod` routine to adjust the coefficients of the left-most GF for several epochs, i.e., complete passes over the data. Similarly, the left chain can be collapsed to form

$$\tilde{\boldsymbol{H}}_{\mathrm{l}} = \boldsymbol{H}_{\mathrm{l}}\boldsymbol{M}, \tag{23}$$

again, due to the linear nature of the model, and use the `descentMethod` routine to adjust now the coefficients of the right-most GF for several epochs. This refitting procedure, consisting of the left and right updates depicted in Fig. 2, is performed for several rounds or until a convergence criterium is met. Due to the collapsing and alternating updates, we refer to this method as staggered RELIEF.

Remark 1: Although, in theory, the linear operators (22) and (23) should be no different than a "layered" implementation, i.e., an implementation where every batch sequentially undergoes the linear operators, we make the reader aware that, at implementation time, there might be differences due to rounding errors and the selected floating-point representation of the involved matrices. Hence, results might slightly vary but not considerable.

B. Some Words About (Full) Backpropagation

A straightforward way to solve (19), including the previously discussed aspects, is *backpropagation* (BP) [65] equipped with a stochastic descent method. Within signal processing and neural networks (NNs), BP is a celebrated method for updating the coefficients of the different layers composing a NN. However, although BP can find the global optimum of (19) for cascaded structures employing C-GFs as modules, based on the result from Theorem 2, for other GF structures, it might incur a high computational cost due to the (possibly) large number of parameters per stage and the full backpropagation of the gradient. Moreover, due to the increased sensitivity of the coefficients, the learning has to be done with a small step size which directly affects the convergence of the method. In NNs, this last issue is

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

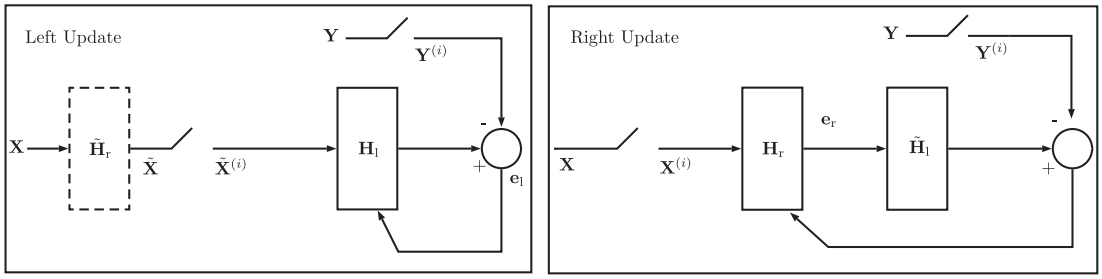
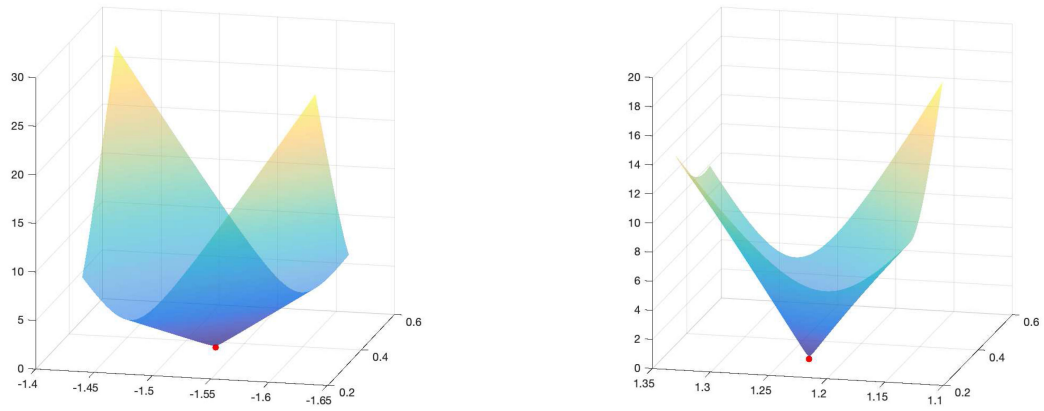


Fig. 2. Schematic of proposed refitting update procedure for the proposed staggered RELIEF. (left) Update cycle for the left coefficients. (right) Update cycle for the right coefficients. Here, $\boldsymbol{X}^{(i)}$ and $\boldsymbol{Y}^{(i)}$ are the ith input- and output-data batches, respectively; and $\boldsymbol{e}_\mathrm{l}$ and $\boldsymbol{e}_\mathrm{r}$ the error signals required for gradient computations.



(a) **(b)**

Fig. 3. Comparison of the error surface profiles of the (a) direct, and (b) cascaded implementation for a C-GF with roots $[0.3, 1.23]$ for a sensor network graph with $N = 32$ nodes.

usually tackled by the nonlinearities and the batch normalization layers present in the architectures, see, e.g., [66], which are not present in the cascaded structure. In addition, differently from the staggered RELIEF, the need to know beforehand the number of modules to use, i.e., Q is a hyper-parameter that needs to be set, and the large number of parameter updates, i.e., matrix-vector products, that have to be made when learning rates (descent steps) are small make this approach not as appealing as the proposed staggered RELIEF. However, despite that we advocate the use of a variant of RELIEF instead of full BP for finding the coefficients of the involved GFs, we do encourage to use full BP steps for fine-tuning after an initial coefficient configuration has been found using the proposed data-driven staggered RELIEF. That is, after an appropriate number of modules, $\hat{Q}$, and their respective coefficients have been *fitted* using the the proposed data-driven staggered RELIEF, several steps of full BP can be employed to improve the fitting quality locally, i.e., further reduction of the fitting cost by local optimization on the neighbourhood of the RELIEF-based solution.

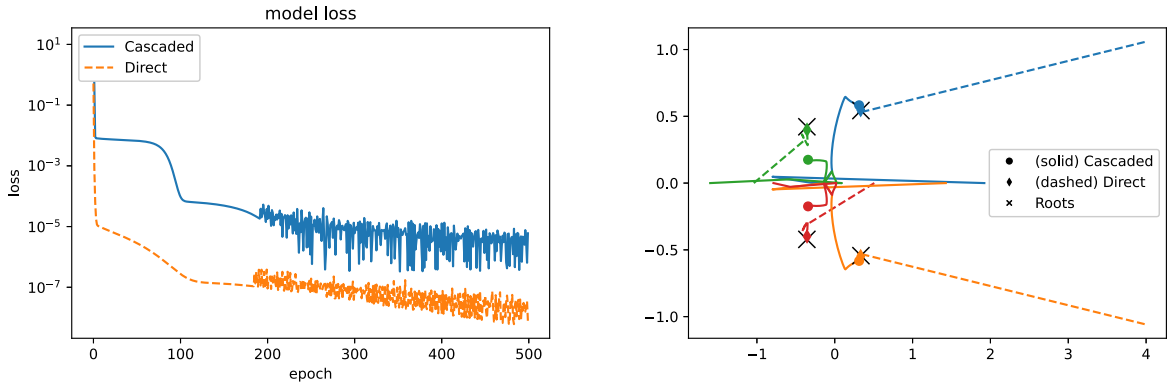
VI. Numerical Results

To illustrate the performance of the introduced cascaded structure, in the following, we present a series of numerical experiments covering the theoretical aspects discussed in the manuscript as well as the model- and data-driven application of the RELIEF.

A. Error Surfaces and Root Analysis

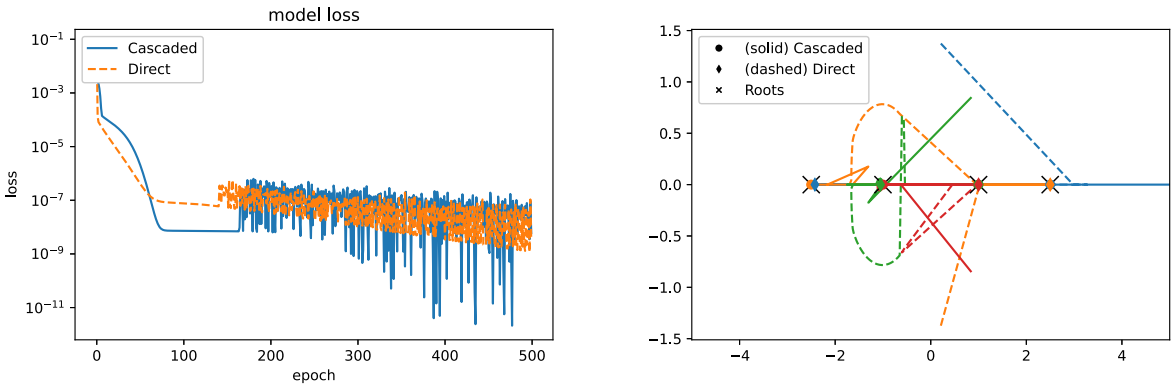
First, we present an example that illustrates the results of Theorems 1 and 2. Here, we consider a simple example for a sensor network with $N = 32$ nodes constructed using the GSPToolbox [67]. The GSO considered in this example is the combinatorial Laplacian of the network. Here, we consider a 2nd-order C-GF with coefficients $\boldsymbol{\theta} = [0.369, -1.53, 1]^\top$. The roots of the corresponding polynomial are $[0.3, 1.23]$. In Fig. 3 , we show the corresponding error surfaces for the different implementations, i.e., (left) direct and (right) cascaded. In this experiment, we generated 1000 Gaussian-distributed graph signals, i.e., $\boldsymbol{X} \in \mathbb{R}^{N\times 1000}$, and filtered them using the above mentioned C-GF. The filtered graph signals are then perturbed with additive white Guassian noise whose standard deviation is $\sigma = 10^{-3}$, thus the observations follow the model $\mathbf{Y} = \mathbf{H}\boldsymbol{X} + \mathbf{N}$, where $\mathbf{N}$ denotes the additive noise and $\mathbf{H}$ the applied C-GF. The error surface shown in Fig. 3 is then the norm of the fitting error, e..g, $\|\mathcal{H}(\boldsymbol{\Theta})\boldsymbol{X} - \mathbf{Y}\|_\mathrm{F}$. While for the direct implementation, the cost is known to be convex [cf. Fig. 3(a)],

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.



(a) (b)

Fig. 4. Comparison of the fitting performance for the direct and cascaded implementation of a C-GF with roots $[0.336 + 0.543i\, 0.336 - 0.543i,\ -0.358 + 0.421i,\ -0.358 - 0.421i]$. (a) Evolution of fitting error per epoch. (b) Trajectory of the estimated filter polynomial roots.



(a) (b)

Fig. 5. Comparison of the fitting performance for the direct and cascaded implementation of a C-GF with roots $[-2.5,\ 2.5,\ -1,\ 1]$. (a) Evolution of fitting error per epoch. (b) Trajectory of the estimated filter polynomial roots.

the minimizer has not changed for the cascaded implementation as discussed in Theorems 1 and 2. Further, the profile of the cost function is also different. That is, although no saddle points were introduced, the gradient towards the minimum in Fig. 3(b) can be seen larger than that of Fig. 3(a). In particular, in one of the dimensions (coefficients) of the direct implementation the cost function exhibits an elongated behavior, i.e., different sensitivity for each coefficient. This simple example, illustrates the theoretical results derived in this work with respect to the cascaded implementation of C-GFs, which are consistent with well-known results within signal processing and adaptive filtering.

Now, we briefly discuss one of the problems of the cascaded implementation, namely, close and mirrored roots. As discussed before, the error surface of the cascaded implementation exhibits amenable properties when there are no repeated roots. However, when the roots are close to each other and mirrored, i.e., pair of complex conjugate roots, the chances that the roots move to the area where a singularity appears, i.e., repeated roots, increases. Thus, the cascaded implementation might exhibit problems finding the appropriate roots in these cases or a slower convergence. To show an example of this behaviour, let us consider a C-GF whose coefficients are $\boldsymbol{\theta} = [5.2,\ 0.23,\ 1.21,\ 0.45,\ 0.65]$. In this case, the roots of the corresponding 4th-order polynomial (up to 3 digits of precision) are $[0.336 + 0.543i\, 0.336 - 0.543i,\ -0.358 + 0.421i,\ -0.358 - 0.421i]$. A comparison of the fitting error between the direct and cascaded implementation, for this case, is shown in Fig. 4. In Fig. 4(a) two things can be observed. First, the direct implementation obtains a better fit (loss) and with a much faster rate. Second, the cascaded implementation enters a *plateau* early on but escapes, i.e., it keeps descending. The plateau is reached when the root estimates collapse on the real axis and have similar values (see Fig. 4(b)). After the roots meet, they move away from the real axis and move towards the respective conjugate pairs. As discussed before, when a stochastic method is used to perform the optimization, the cascaded implementation is able to escape from saddle points without much trouble. Notice here that while at the 500th iteration, the direct implementation is really close to the original roots, the cascaded implementation still requires more iterations to reach them. In contrast with this result, we have the experiment shown in Fig. 5. In this case, another C-GF with roots $[-2.5,\ 2.5,\ -1,\ 1]$ is identified through data. Differently from the previous case, the cascaded implementation avoids the collapse of the roots as seen in Fig. 5(b) leading to a better fitting than the direct implementation and at a faster

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

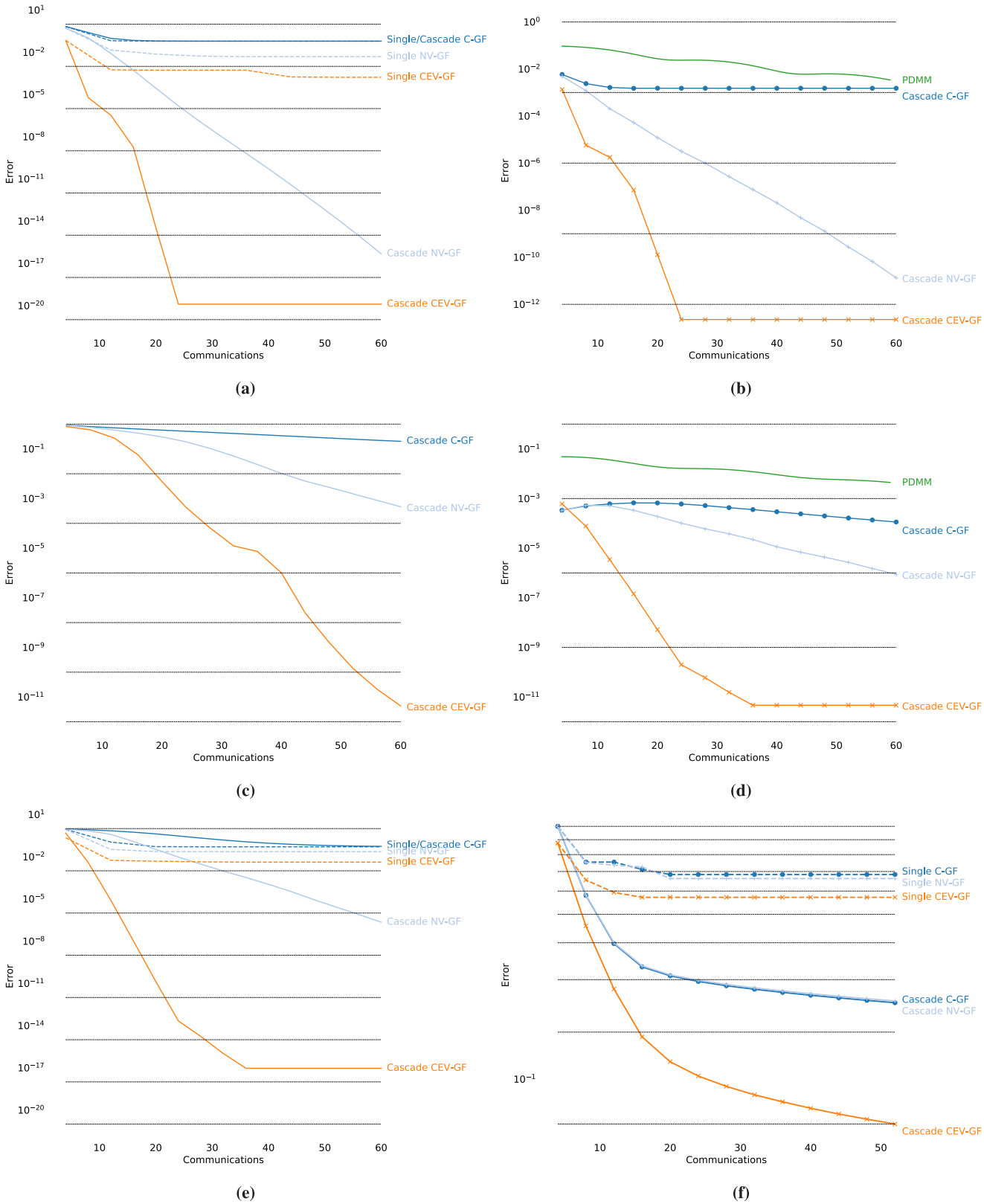


Fig. 6. (Left) Model error comparison, i.e., $\|\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta}) - \boldsymbol{H}^*\|_F^2$, of direct and cascaded GF implementations for (a) direct solver for $N = 100$, (c) operator-based (no-explicit system matrix) solver for $N = 400$ and (e) LSMR solver with explicit system matrix and diagonal preconditioning (Jacobi) for $N = 500$. (Right) Action error comparison, i.e., $\mathbb{E}\{\|\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta})\boldsymbol{x} - \boldsymbol{H}^*\boldsymbol{x}\|_2^2\}$, for different GFs and PDMM for (b) $N = 100$ and (d) $N = 500$. (f) Error comparison for single and cascaded GF implementation optimized with GCNN framework.

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

rate, see Fig. 5(a). Thus, this illustrates that when areas near singularities are not reached by the root estimates, the cascaded implementation can outperform the fitting of the direct implementation even for relatively low GF orders. For a fair comparison, in these examples, we have employed a common rule for the step size of the stochastic gradient descent in both cases, particularly $\delta_k = 2/(k+3)$, where k denotes the epoch. A more careful selection of the step size rule could have reduced the jitter observed in Fig. 4, however, the trend is clear from the plot. Here, we have used a `batchsize`$= 2000$ and a sensor network with $N = 64$ nodes. The GSO for constructing these C-GFs was the normalized Laplacian of the network, and the number of samples used for fitting the models was 132 000.

B. RELIEF Application: Distributed Consensus

To demonstrate the performance of the model-driven RELIEF method, we study the problem of consensus over networks [68]. That is, given a network (graph), we desire to compute the average of the signal over the network, i.e.,

$$\boldsymbol{y} = \frac{1}{N}\mathbf{1}\mathbf{1}^T\boldsymbol{x} = \boldsymbol{H}^*\boldsymbol{x}, \tag{24}$$

where $\mathbf{1}$ is the N-dimensional all-one vector. To do so, we explore the proposed implementations on different network sizes and base GFs, i.e., C-GF, NV-GF, and CEV-GF. In these experiments, for small problem sizes, i.e., $N < 300$, we employ direct solvers while for larger problem sizes LSMR based solvers. In particular, for $N = 400$ a solver with a non-explicit system matrix has been implemented. For $N = 500$ a standard LSMR solver, with explicit system matrix and diagonal preconditioning, has been used.

In Fig. 6(a), Fig. 6(c) and Fig. 6(e) a comparison for respectively $N = 100$, $N = 400$ and $N = 500$ in terms of model error, i.e., $\|\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta}) - \boldsymbol{H}^*\|_F^2$, is shown for different types of GFs and communication exchanges, i.e., data exchanges between neighbors. Here, the name *direct* refers to a direct implementation, i.e., the order of the filter is equal to the number of communication rounds, while the name *cascaded* to the proposed cascaded implementation. In all these figures, we observe that the cascaded implementation outperforms the direct implementation, and the best performance in terms of error is achieved by the CEV-GF.

To further test the proposed implementation, we compare the trained cascaded GFs with the primal-dual method of multipliers (PDMM)[4] [69] for network consensus. In Fig. 6(b) and Fig. 6(d), we observe that the convergence speed of PDMM (in terms of communication rounds) is not comparable with the one obtained by the cascaded CEV-GF, despite that PDMM eventually guarantees consensus. This implies that by an adequate selection of the weight matrices, consensus can be achieved (up-to machine precision) in a low number of steps. Here, the reported error is the *action* error, i.e., $\|\mathcal{H}(\boldsymbol{S};\boldsymbol{\Theta})\boldsymbol{x} - \boldsymbol{H}^*\boldsymbol{x}\|_2^2$.

Finally, we perform a test illustrating the performance of the data-driven staggered RELIEF using the GCNN framework for coefficient learning. In this example, we fit the same consensus operator on a community network with $N = 200$ nodes. For the GCNN-based solver, the number of filters (layers) has been set to $Q = 15$, and a set of $M = 10^3$ randomly generated data pairs have been generated. The batch size has been set to 3200 and the learning rate to 0.001. In Fig. 6(f), the results for this experiment are shown. Similar as in the other experiments, we observe that the cascaded implementation outperforms the direct implementation, and that among all the types of GFs, the CEV-GF obtains the best performance over a randomly generated test set. Here, as the GCNN framework is data dependent, the error measure reported is again the action error.

[4]PDMM is an alternative distributed optimization tool to the classical alternating direction method of multipliers (ADMM), which is often characterized by a faster convergence.

VII. Concluding Remarks

In this work, we have introduced a cascaded implementation of generalized graph filters and we have drawn connections with neural network architectures for graph-supported data. Furthermore, we proposed an efficient stable algorithm for designing/learning the GF coefficients, which can be implemented leveraging state-of-the-art sparse solvers and preconditioning techniques in the model-driven case. In addition, by minor modifications, we showed that the proposed method can be applied in the data-driven setting as well and that the deep learning framework can be combined with the proposed algorithm to fit the involved GFs when data is available or easy to generate. Finally, we illustrated the applicability of the proposed implementations and design algorithms in a network consensus application, where we have shown that CEV-GF filters achieve machine-precision accuracy at a significantly lower communication cost than existing methods.

Appendix

A. Proof Lemma 1

To prove the result, we first recall that

$$\frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial h(\boldsymbol{\Theta})}{\partial \boldsymbol{\theta}} \tag{25}$$

$$= \frac{\partial h(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}}\frac{\partial \boldsymbol{\Theta}}{\partial \boldsymbol{\theta}}, \tag{26}$$

where $\partial f(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$, $\partial h(\boldsymbol{\Theta})/\partial\boldsymbol{\theta}$ and $\partial\boldsymbol{\Theta}/\partial\boldsymbol{\theta}$ are the function gradients and the parameter Jacobian matrix, respectively, with respect to $\boldsymbol{\theta}$. Now, assume that $\boldsymbol{\Theta}^*$ is a critical point of $h(\boldsymbol{\Theta})$, i.e.,

$$\left.\frac{\partial h(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}}\right|_{\boldsymbol{\Theta}^*} = 0. \tag{27}$$

By assumption, there exists a $\boldsymbol{\theta}^*$ such that $\varphi(\boldsymbol{\theta}^*) = \boldsymbol{\Theta}^*$ (due to the existence and surjectiveness of φ). Hence, by (25), $\boldsymbol{\theta}^*$ is a critical point of $f(\boldsymbol{\theta})$ and has the same nature as that of $\boldsymbol{\Theta}^*$ due to

$$\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}^*} = \left(\frac{\partial \boldsymbol{\Theta}}{\partial \boldsymbol{\theta}}\right)^{\mathrm{H}}\Bigg|_{\boldsymbol{\Theta}^*} \nabla^2_{\boldsymbol{\Theta}} h(\boldsymbol{\Theta})\big|_{\boldsymbol{\Theta}^*}\left(\frac{\partial \boldsymbol{\Theta}}{\partial \boldsymbol{\theta}}\right)\Bigg|_{\boldsymbol{\Theta}^*}. \tag{28}$$

In the case that there exists a critical point of $h(\boldsymbol{\Theta})$, $\varphi(\boldsymbol{\theta}^+) = \boldsymbol{\Theta}^+$, such that $\boldsymbol{\theta}^+$ is not a critical point itself, i.e.,

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

$\partial f(\boldsymbol{\theta}^+)/\partial\boldsymbol{\theta}|_{\boldsymbol{\theta}^+} \neq 0$, then φ is not differentiable at $\boldsymbol{\theta}^+$ as the equivalency property (25) does not hold.

As the nature of a critical point as $\boldsymbol{\Theta}^*$ does not change with respect to that of $\boldsymbol{\theta}^*$, we now are left to show the nature of $\boldsymbol{\Theta}^+$. First, let us consider an open ball centered at $\boldsymbol{\theta}^+$ with radio $r > 0$, i.e., $\mathcal{B}_{\boldsymbol{\theta}^+,r}$. Since $\boldsymbol{\theta}^+$ is not a critical point, there exists a direction $\Delta\boldsymbol{\theta}$ where $f(\boldsymbol{\theta}^+ + \Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta}^+)$ and $\boldsymbol{\theta}^+ + \Delta\boldsymbol{\theta} \in \mathcal{B}_{\boldsymbol{\theta}^+,r}$. Further, by hypothesis, φ is continuous, then $\varphi(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \in \mathcal{B}'_{\boldsymbol{\Theta}^+,r}$, where $\mathcal{B}'_{\boldsymbol{\Theta}^+,r}$ is the image $\mathcal{B}_{\theta^+,r}$ under φ. Hence, $h(\varphi(\boldsymbol{\theta}^+ + \delta\boldsymbol{\theta})) < \varphi(\boldsymbol{\Theta}^+)$, which implies the result of the Lemma.

B. Proof of Theorem 1

To show this, we build intuition for $Q = 2$ using modules with structure

$$\boldsymbol{H}_q = \boldsymbol{I} + \boldsymbol{\Phi}_1^{(q)}\boldsymbol{S}. \tag{29}$$

They can be seen as the simplest modules to implement, i.e., root expansion.

The overall linear operator implemented using this construction is

$$\begin{aligned}\boldsymbol{H}_2\boldsymbol{H}_1 &= (\boldsymbol{I} + \boldsymbol{\Phi}_1^{(2)}\boldsymbol{S})(\boldsymbol{I} + \boldsymbol{\Phi}_1^{(1)}\boldsymbol{S}) \\ &= \boldsymbol{I} + [\boldsymbol{\Phi}_1^{(2)} + \boldsymbol{\Phi}_1^{(1)}]\boldsymbol{S} + \boldsymbol{\Phi}_1^{(2)}\boldsymbol{S}\boldsymbol{\Phi}_1^{(1)}\boldsymbol{S},\end{aligned} \tag{30}$$

which is not a CEV-GF. Hence, no surjective map is available.

Now, let us equip with the assumption that the GF coefficient matrices commute with the GSO, i.e., $\boldsymbol{\Phi}_i^{(q)}\boldsymbol{S} = \boldsymbol{S}\boldsymbol{\Phi}_i^{(q)}\ \forall\, i \in [K],\ q \in [Q]$. Then, the expression in (30) can be rewritten as

$$\boldsymbol{H}_2\boldsymbol{H}_1 = \boldsymbol{I} + [\boldsymbol{\Phi}_1^{(2)} + \boldsymbol{\Phi}_1^{(1)}]\boldsymbol{S} + \tilde{\boldsymbol{\Phi}}\boldsymbol{S}^2. \tag{31}$$

Unfortunately, $\tilde{\boldsymbol{\Phi}}$, in general, does not share the support with $\boldsymbol{S}$, hence (31) is, again, not a CEV-GF. As these observations carry on for the NV-GFs and CEV-GFs with different module orders, we can ensure that the φ mapping does not exist for these structures.

However, when $\boldsymbol{\Phi}_i^{(q)} = \phi_i^{(q)}\boldsymbol{I}\ \forall, i \in [K],\ q \in [Q]$, it is possible to show that the mapping φ exists as in this case $\tilde{\boldsymbol{\Phi}}$ in (31) is another scaled identity matrix, implying that the resulting operator is a C-GF, and thus that the map is onto (surjective). Finally, the map is continuous as the roots of a polynomial (cascaded form) depend continuously on the coefficients (direct form).

C. Proof of Theorem 2

To show this result, we first recall

$$\frac{\partial h(\boldsymbol{\Theta})}{\partial\boldsymbol{\Theta}} = \frac{\partial f(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\frac{\partial\boldsymbol{\theta}}{\partial\boldsymbol{\Theta}}. \tag{32}$$

Using this relation, a saddle point in (6) implies that the Jacobian $\partial\boldsymbol{\theta}/\partial\boldsymbol{\Theta}$ is singular. So, to provide the result of the proposition, in the following, we show the conditions in which the Jacobian has a zero determinant.

Due to the particular cascaded implementation, we introduce the notation $\alpha_q = \phi_1^{(q)}$ for simplicity. Therefore, the Jacobian $\boldsymbol{J} := \partial\boldsymbol{\theta}/\partial\boldsymbol{\Theta}$ has entries

$$[\boldsymbol{J}]_{i,j} = \partial\phi_i/\partial\alpha_j. \tag{33}$$

Taking the partial derivative w.r.t α_j of $\mathcal{H}$, we obtain

$$\frac{\partial\mathcal{H}}{\partial\alpha_j} = \boldsymbol{S}\prod_{q=1,\ q\neq j}^{K}(\boldsymbol{I} + \alpha_q\boldsymbol{S}). \tag{34}$$

Doing the same for $\boldsymbol{H}$, we obtain

$$\frac{\boldsymbol{H}}{\partial\alpha_j} = \sum_{k=1}^{K}\frac{\partial\phi_k}{\partial\alpha_j}\boldsymbol{S}^k. \tag{35}$$

As by hypothesis, $\mathcal{H}$ and $\boldsymbol{H}$ implements the same C-GF, we can equate the terms with the same order, i.e., power of $\boldsymbol{S}$, and obtain the following relations for the entries of the Jacobian, i.e.,

$$J_{1,j} := [\boldsymbol{J}]_{1,j} = \frac{\partial\phi_1}{\partial\alpha_j} = 1,\ \forall\, j \tag{36}$$

$$J_{2,j} := [\boldsymbol{J}]_{2,j} = \frac{\partial\phi_2}{\partial\alpha_j} = \sum_{q_1\neq j}^{K}\alpha_{q_1} \tag{37}$$

$$\vdots \tag{38}$$

$$J_{i,j} := [\boldsymbol{J}]_{i,j} = \frac{\partial\phi_i}{\partial\alpha_j} = \sum_{\substack{q_1<\ldots<q_{i-1}\\ q_1,\ldots,q_{i-1}\neq j}}^{K}\prod_{l=1}^{i-1}\alpha_{q_l}. \tag{39}$$

Now, the only thing left is to find the determinant of $\boldsymbol{J}$. First, let us consider the Jacobian matrix

$$\boldsymbol{J} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ J_{2,1} & J_{2,2} & \cdots & J_{K,2} \\ \vdots & \vdots & \vdots & \vdots \\ J_{K,1} & J_{K,2} & \cdots & J_{K,2}. \end{bmatrix} \tag{40}$$

As row substraction does not change the determinant of $\boldsymbol{J}$, we now reduce $\boldsymbol{J}$ by a series of row operations to obtain a closed form for its determinant, similar to the one obtain for matrices derived for time-domain IIR filters in [44].

Let us subtract the first column from every other column in $\boldsymbol{J}$, i.e.,

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ J_{2,1} & (\alpha_1-\alpha_2) & \cdots & (\alpha_1-\alpha_K) \\ \vdots & \vdots & \vdots & \\ J_{k,1} & (\alpha_1-\alpha_2)J^{(1)}_{k-1,1} & \cdots & (\alpha_1-\alpha_K)J^{(1)}_{k-1,K-1} \\ \vdots & \vdots & \vdots & \\ J_{K,1} & (\alpha_1-\alpha_2)J^{(1)}_{K-1,1} & \cdots & (\alpha_1-\alpha_K)J^{(1)}_{K-1,K-1} \end{bmatrix} \tag{41}$$

where

$$[\boldsymbol{J}^{(1)}]_{i,j} = J^{(1)}_{i,j} := \sum_{\substack{q_1<\ldots<q_{i-1}\\ q_1,\ldots,q_{i-1}\neq 1\neq j+1}}^{K}\prod_{l=1}^{i-1}\alpha_{q_l} \tag{42}$$

and $[\boldsymbol{J}^{(1)}]_{1,j} = 1\ \forall\, j$.

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

Using the property of the determinant for block matrices, and the equivalence $\det(\boldsymbol{J}) = \det(\boldsymbol{J}_1)$, we factorize the determinant of $\boldsymbol{J}$ as

$$\det(\boldsymbol{J}) = \prod_{k=2}^{K} (\alpha_1 - \alpha_k) \det(\boldsymbol{J}^{(1)}). \tag{43}$$

Applying this process recursively, i.e., row subtraction and determinant factorization, we can keep reducing the determinant expression until we get

$$\det(\boldsymbol{J}) = \prod_{\substack{i,j=1 \\ i<j}}^{K} (\alpha_i - \alpha_j). \tag{44}$$

Thus, the above expression implies the result of the theorem.

D. LSMR Algorithm

Algorithm 3: LSMR Algorithm for solving $\boldsymbol{Ax} \approx \boldsymbol{b}$.

Result: $\boldsymbol{x}_k$: least squares solution
Input: $\boldsymbol{A}$, $\boldsymbol{b}$, ϵ_{tol}
initialization: $\beta_1 \boldsymbol{u}_1 = \boldsymbol{b}$, $\alpha_1 \boldsymbol{v}_1 = \boldsymbol{A}^T \boldsymbol{u}_1$, $\bar{\alpha}_1 = \alpha_1$,
($\beta_1 \boldsymbol{u}_1 = \boldsymbol{b}$ is shorthand for $\beta_1 = \|\boldsymbol{b}\|$, $\boldsymbol{u}_1 = \boldsymbol{b}/\beta_1$)
$\bar{\zeta}_1 = \alpha_1 \beta_1$, $\rho_0 = 1$, $\bar{\rho}_0 = 1$, $\bar{c}_0 = 1$, $\bar{s}_0 = 0$,
$\boldsymbol{h}_1 = \boldsymbol{v}_1$, $\bar{\boldsymbol{h}}_0 = \boldsymbol{0}$, $\boldsymbol{x}_0 = \boldsymbol{0}$;
for $k = 1, 2, \ldots,$ *until convergence* **do**
  :: Bidiagonalization ::
  $\beta_{k+1} \boldsymbol{u}_{k+1} = \boldsymbol{A} \boldsymbol{v}_k - \alpha_k \boldsymbol{u}_k$;
  $\alpha_{k+1} \boldsymbol{v}_{k+1} = \boldsymbol{A}^T \boldsymbol{u}_{k+1} - \beta_{k+1} \boldsymbol{v}_k$;
  :: Rotation for first QR factorization ::
  $\rho_k = (\bar{\alpha}_k^2 + \beta_{k+1}^2)^{\frac{1}{2}}$;
  $c_k = \bar{\alpha}_k / \rho_k$;
  $s_k = \beta_{k+1} / \rho_k$;
  $\theta_{k+1} = s_k \alpha_{k+1}$;
  $\bar{\alpha}_{k+1} = c_k \alpha_{k+1}$;
  :: Rotation for second QR factorization ::
  $\bar{\theta}_k = \bar{s}_{k-1} \rho_k$;
  $\bar{\rho}_k = ((\bar{c}_{k-1} \rho_k)^2 + \theta_{k+1}^2)^{\frac{1}{2}}$;
  $\bar{c}_k = \bar{c}_{k-1} \rho_k / \bar{\rho}_k$;
  $\bar{s}_k = \theta_{k+1} / \bar{\rho}_k$;
  $\zeta_k = \bar{c}_k \bar{\zeta}_k$;
  $\bar{\zeta}_{k+1} = -\bar{s}_k \bar{\zeta}_k$;
  :: Solution Update ::
  $\bar{\boldsymbol{h}}_k = \boldsymbol{h}_k - (\bar{\theta}_k \rho_k / (\rho_{k-1} \bar{\rho}_{k-1})) \bar{\boldsymbol{h}}_{k-1}$;
  $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + (\zeta_k / (\rho_k \bar{\rho}_k)) \bar{\boldsymbol{h}}_k$;
  $\boldsymbol{h}_{k+1} = \boldsymbol{v}_{k+1} - (\theta_{k+1} / \rho_k) \boldsymbol{h}_k$;
  :: Convergence Check ::
  if $|\bar{\zeta}_{k+1}| < \epsilon_{\text{tol}}$ **then**
    stop;
  end
end

REFERENCES

[1] M. Coutino and G. Leus, "On distributed consensus by a cascade of generalized graph filters," in *Proc. 53rd Asilomar Conf. Signals, Syst., Comput.*, 2019, pp. 46–50.

[2] S. K. Narang, A. Gadde, and A. Ortega, "Signal processing techniques for interpolation in graph structured data," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2013, pp. 5445–5449.

[3] A. Sandryhaila and J. M. Moura, "Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 80–90, Sep. 2014.

[4] C. Hu, J. Sepulcre, K. A. Johnson, G. E. Fakhri, Y. M. Lu, and Q. Li, "Matched signal detection on graphs: Theory and application to brain imaging data classification," *NeuroImage*, vol. 125, pp. 587–600, 2016.

[5] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[6] G. Taubin, "Geometric signal processing on polygonal meshes," in *Proc. Eurographics*, 2000.

[7] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *J. Mach. Learn. Res.*, vol. 7, no. 11, pp. 2399–2434, 2006.

[8] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[9] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2320–2333, May 2019, doi: 10.1109/TSP.2019.2904925.

[10] P. Di Lorenzo, S. Barbarossa, P. Banelli, and S. Sardellitti, "Adaptive least mean squares estimation of graph signals," *IEEE Trans. Signal Inf. Process. Over Netw.*, vol. 2, no. 4, pp. 555–568, Dec. 2016.

[11] A. Loukas, M. Zuniga, I. Protonotarios, and J. Gao, "How to identify global trends from local decisions? Event region detection on mobile networks," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 1177–1185.

[12] L. F. Chamon and A. Ribeiro, "Finite-precision effects on graph filters," in *Proc. 5th IEEE Glob. Conf. Signal Inf. Process.*, 2017, pp. 603–607.

[13] T. Aittomäki and G. Leus, "Graph filter design using sum-of-squares representation," in *Proc. 27th Eur. Signal Process. Conf.*, 2019, pp. 1–5.

[14] D. I. Shuman, P. Vandergheynst, and P. Frossard, "Distributed signal processing via Chebyshev polynomial approximation," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 4, no. 4, pp. 736–751, Dec. 2018, doi: 10.1109/TSIPN.2018.2824239.

[15] S. Segarra, A. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. Signal Process*, vol. 65, no. 15, pp. 4117–4131, Aug. 2017.

[16] M. Coutino, E. Isufi, and G. Leus, "Distributed edge-variant graph filters," in *Proc. IEEE 7th Int. Workshop Comp. Adv. Multi-Sensor Adap. Process.*, 2017, pp. 1–5.

[17] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Philadelphia, PA, USA: SIAM, 2003, vol. 82.

[18] S. Segarra, G. Mateos, A. G. Marques, and A. Ribeiro, "Blind identification of graph filters," *IEEE Trans. Signal Process.*, vol. 65, no. 5, pp. 1146–1159, Mar. 2016.

[19] M. H. Hayes, *Statistical Digital Signal Processing and Modeling*. Hoboken, NJ, USA: Wiley, 2009.

[20] A. H. Sayed, *Fundamentals of Adaptive Filtering*. Hoboken, NJ, USA: Wiley, 2003.

[21] P. Di Lorenzo, P. Banelli, E. Isufi, S. Barbarossa, and G. Leus, "Adaptive graph signal processing: Algorithms and optimal sampling strategies," *IEEE Trans. Signal Process.*, vol. 66, no. 13, pp. 3584–3598, Jul. 2018.

[22] C. Cowan, "Performance comparisons of finite linear adaptive filters," in *IEE Proc. F (Commun., Radar Signal Process.)*, vol. 134, no. 3, 1987, pp. 211–216.

[23] J. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 2, pp. 304–337, Apr. 1984.

[24] A. Uncini, *Fundamentals of Adaptive Signal Processing*. Berlin, Germany: Springer, 2015.

[25] P. Prandoni and M. Vetterli, "An FIR cascade structure for adaptive linear prediction," *IEEE Trans. Signal Process.*, vol. 46, no. 9, pp. 2566–2571, Sep. 1998.

[26] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. Int. Conf. Learn. Representations*, CBLS, Apr. 2014. [Online]. Available: http://openreview.net/document/d332e77d- 459a-4af8-b3ed-55ba9662182c

[27] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://tensorflow.org/

[28] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://keras.io

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.

[29] N. J. Higham, *Functions of Matrices: Theory and Computation*. Philadelphia, PA, USA: SIAM, 2008, vol. 104.
[30] M. Coutino, E. Isufi, and G. Leus, "Distributed edge-variant graph filters," in *Proc. Int. Workshop Comp. Adv. Multi-Sensor Adaptive Process.*, 2017, pp. 1–5.
[31] E. Isufi, P. Banelli, P. Di Lorenzo, and G. Leus, "Observing and tracking bandlimited graph processes from sampled measurements," *Signal Process.*, vol. 177, 2020, Art. no. 107749, doi: 10.1016/j.sigpro.2020.107749.
[32] W. Huang, L. Goldsberry, N. F. Wymbs, S. T. Grafton, D. S. Bassett, and A. Ribeiro, "Graph frequency analysis of brain signals," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 7, pp. 1189–1203, Oct. 2016.
[33] M. Sun, E. Isufi, N. M. de Groot, and R. C. Hendriks, "Graph-time spectral analysis for atrial fibrillation," *Biomed. Signal Process. Control*, vol. 59, 2020, Art. no. 101915.
[34] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018.
[35] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural networks architectures for signals supported on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 1034–1049, Feb. 2019, doi: 10.1109/TSP.2018.2887403.
[36] F. Gama, A. Ribeiro, and J. Bruna, "Stability of graph scattering transforms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8036–8046.
[37] A. N. Nikolakopoulos, D. Berberidis, G. Karypis, and G. B. Giannakis, "Personalized diffusions for top-n recommendation," in *Proc. 13th ACM Conf. Recommender Syst.*, 2019, pp. 260–268.
[38] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Trans. Signal Process*, vol. 65, no. 2, pp. 274–288, Jan. 2017.
[39] J. Liu, E. Isufi, and G. Leus, "Filter design for autoregressive moving average graph filters," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 5, no. 1, pp. 47–60, Mar. 2019, doi: 10.1109/TSIPN.2018.2854627.
[40] L. B. Saad, B. Beferull-Lozano, and E. Isufi, "Quantization analysis and robust design for distributed graph filters," *IEEE Trans. Signal Process.*, vol. 70, pp. 643–658, 2022, doi: 10.1109/TSP.2021.3139208.
[41] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*. Boca Raton, FL, USA: CRC Press, 2002.
[42] B. D. Rao, "Adaptive iir filtering using cascade structures," in *Proc. 27th Asilomar Conf. Signals, Syst. Comput.*, 1993, pp. 194–198.
[43] J. O. Smith, *Introduction to Digital Filters with Audio Applications*, W3K Publishing, 2007. [Online]. Available: http://books.w3k.org/
[44] M. Nayeri and W. K. Jenkins, "Alternate realizations to adaptive IIR filters and properties of their performance surfaces," *IEEE Trans. Circuits Syst.*, vol. 36, no. 4, pp. 485–496, Apr. 1989.
[45] T. Söderström and P. Stoica, "Some properties of the output error method," *Automatica*, vol. 18, no. 1, pp. 93–99, 1982.
[46] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020, doi: 10.1016/j.aiopen.2021.01.001.
[47] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*, 2010, pp. 177–186.
[48] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[49] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.
[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[51] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations*, Toulon, France: OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl
[52] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871.
[53] J. Li and P. Stoica, "Efficient mixed-spectrum estimation with applications to target feature extraction," *IEEE Trans. Signal Process.*, vol. 44, no. 2, pp. 281–295, Feb. 1996.
[54] D.C.-L. Fong and M. Saunders, "LSMR: An iterative algorithm for sparse least-squares problems," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2950–2971, 2011.
[55] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. Math. Softw.*, vol. 8, no. 1, pp. 43–71, 1982.
[56] Y. Xu and W. Yin, "A block coordinate descent method for regularized Multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM J. Imag. Sci.*, vol. 6, no. 3, pp. 1758–1789, 2013.
[57] L. Grippo and M. Sciandrone, "On the convergence of the block nonlinear gauss-seidel method under convex constraints," *Operations Res. Lett.*, vol. 26, no. 3, pp. 127–136, 2000.
[58] G. B. Giannakis, Y. Shen, and G. V. Karanikolas, "Topology identification and learning over graphs: Accounting for nonlinearities and dynamics," *Proc. IEEE*, vol. 106, no. 5, pp. 787–807, May 2018.
[59] M. Coutino, E. Isufi, T. Maehara, and G. Leus, "State-space network topology identification from partial observations," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 6, pp. 211–225, Feb. 2020.
[60] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 16–43, May 2019.
[61] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from filtered signals: Graph system and diffusion Kernel identification," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 5, no. 2, pp. 360–374, Jun. 2019.
[62] T. Tieleman and G. Hinton, "Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
[63] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, CA, USA, 2015. [Online]. Available: https://dblp.org/rec/journals/corr/KingmaB14.bib
[64] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," 2018. [Online]. Available: https://openreview.net/forum?id=B1Yy1BxCZ
[65] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[66] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[67] N. Perraudin *et al.*, "GSPBOX: A toolbox for signal processing on graphs," Aug. 2014, *arXiv:1408.5781*.
[68] T. Li and J.-F. Zhang, "Consensus conditions of multi-agent systems with time-varying topologies and stochastic communication noises," *IEEE Trans. Autom. Control*, vol. 55, no. 9, pp. 2043–2057, Sep. 2010.
[69] G. Zhang and R. Heusdens, "Distributed optimization using the primal-dual method of multipliers," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 4, no. 1, pp. 173–187, Mar. 2018.

Mario Coutino (Member, IEEE) received the M.Sc. and Ph.D degrees (cum laude) in electrical engineering from the Delft University of Technology, Delft, The Netherlands, in July 2016 and April 2021, respectively. Since October 2020, he has been with TNO, The Netherlands, in the Radar Technology Department, as a Signal Processing Researcher. He has held positions with Thales Netherlands, in 2015, and Bang & Olufsen, during 2015–2016. He was a Visiting Researcher with RIKEN AIP and the Digital Technological Center, University of Minnesota, in 2018 and 2019, respectively. His research interests include array signal processing, signal processing on networks, submodular and convex optimization, and numerical linear algebra. He was the recipient of the Best Student Paper Award for his publication at the CAMSAP 2017 conference in Curacao.

Geert Leus (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from KU Leuven, Belgium, in June 1996 and May 2000, respectively. He is currently a Full Professor with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands. He is a Fellow of EURASIP. He was a Member-at-Large of the Board of Governors of the IEEE Signal Processing Society, the Chair of the IEEE Signal Processing for Communications and Networking Technical Committee, and the Editor-in-Chief of the *EURASIP Journal on Advances in Signal Processing*. He is currently the Chair of the EURASIP Technical Area Committee on Signal Processing for Multisensor Systems and the Editor-in-Chief of the EURASIP Signal Processing. He was the recipient of the 2021 EURASIP Individual Technical Achievement Award, the 2005 IEEE Signal Processing Society Best Paper Award, and the 2002 IEEE Signal Processing Society Young Author Best Paper Award.

Authorized licensed use limited to: TU Delft Library. Downloaded on August 26,2022 at 12:57:19 UTC from IEEE Xplore. Restrictions apply.