EINDHOVEN UNIVERSITY OF TECHNOLOGY

Decoupling multivariate functions using a nonparametric filtered tensor decomposition

Citation for published version (APA):
Decuyper, J., Tiels, K., Weiland, S., Runacres, M. C., & Schoukens, J. (2022). Decoupling multivariate functions using a nonparametric filtered tensor decomposition. *Mechanical Systems and Signal Processing*, *179*, Article 109328. https://doi.org/10.1016/j.ymssp.2022.109328

DOI:
10.1016/j.ymssp.2022.109328

Document status and date:
Published: 01/11/2022

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Feb. 2024

Decoupling multivariate functions using a nonparametric filtered tensor decomposition

Jan Decuyper[a,*], Koen Tiels[b], Siep Weiland[b], Mark C. Runacres[a], Johan Schoukens[a,b]

[a] *Vrije Universiteit Brussel, Pleinlaan 2, 1050, Brussels, Belgium*
[b] *Eindhoven University of Technology, Eindhoven, The Netherlands*

Abstract

Multivariate functions emerge naturally in a wide variety of data-driven models. Popular choices are expressions in the form of basis expansions or neural networks. While highly effective, the resulting functions tend to be hard to interpret, in part because of the large number of required parameters. Decoupling techniques aim at providing an alternative representation of the nonlinearity. The so-called decoupled form is often a more efficient parameterisation of the relationship while being highly structured, favouring interpretability. In this work two new algorithms, based on filtered tensor decompositions of first order derivative information are introduced. The method returns nonparametric estimates of smooth decoupled functions. Direct applications are found in, i.a. the fields of nonlinear system identification and machine learning.

Keywords: Decoupling multivariate functions, Filtered Tensor Decomposition (FTD), Jacobian tensor, Neural Network reduction

1. Introduction

Nonlinear function regression is an essential part of most nonlinear modelling techniques. Be it as an end product, or as an internal element of a dynamical model, a multivariate function is typically used to describe the nonlinear relationship amongst variables. In many applications, a data-driven approach is preferred, e.g. when the required knowledge to build a model from first-principles is unavailable, or when the model is only required to connect the dots, without revealing any potential insight into the relationship. Relying exclusively on data, also known as black-box modelling, entails that the required model-complexity is a priori unknown. Powerful regression techniques are therefore characterised by strong approximation properties. Popular choices are, for instance, neural networks, belonging to the class of universal approximators [1, 2], and all sorts of basis expansions, given the resulting ease in parameter estimation when the basis functions are fixed [3]. Inherent to both types is the tendency to result in large models, described by a large number of parameters. It is important to realise that distinct reasons lie at the origin of the bulky size of both types:

☆This work was supported by the Flemish fund for scientific research FWO under license number G0068.18N.
*Corresponding author
Email address: jan.decuyper@vub.be (Jan Decuyper)

- **Networks** rely on width and/or depth to allow complexity to build. It is only by invoking the power of large numbers that seemingly simple neurons transcend their individual capabilities [4].

- **Basis expansions**, on the other hand, may feature a set of functions of which the complexity can vary. In this case, however, the basis remains fixed while there is no evidence that a sparse representation of the relationship can be obtained given the proposed basis. Related so-called dictionary methods [5] rely on regularisation to enforce sparseness, neglecting the root problem, which is an unfavourable basis.

These insights have led to the development of so-called decoupling techniques. Given a generic continuous real multivariate nonlinear function[1]

$$\boldsymbol{q} = \boldsymbol{f}(\boldsymbol{p}), \tag{1}$$

with $\boldsymbol{q} \in \mathbb{R}^n$ and $\boldsymbol{p} \in \mathbb{R}^m$, the prime objective of a decoupling technique is to introduce an appropriate linear transformation of $\boldsymbol{p}$, denoted $\boldsymbol{V}$. The goal of this transformation is to retrieve an alternative basis in which the nonlinear relationship may be described by a minimal set of univariate continuous real functions, also called *branches*. What adds to the power of the new representation is that the functions, which make up the branches, are not fixed beforehand. An appropriate basis together with matching branch functions is the joint product of the decoupling procedure. The decoupled structure is then of the following form

$$\boldsymbol{f}(\boldsymbol{p}) = \boldsymbol{W}\boldsymbol{g}(\boldsymbol{V}^\top \boldsymbol{p}), \tag{2}$$

where the ith function is $g_i(z_i)$ with $z_i = \boldsymbol{v}_i^\top \boldsymbol{p}$, emphasising that all functions are strictly univariate. A schematic representation of a classical multivariate function (hereafter referred to as coupled function) and a decoupled function are presented in the top section of Fig. 1. The decoupled form has a number of attractive features: relying exclusively on univariate functions, a very structured representation is obtained. Moreover, the one-dimensional functions are much more tractable, which may lead to insight into the relationship. Examples of structure retrieval in dynamical models may be found in [6, 7]. Additionally, decoupled functions tend to be a much more efficient representation of the nonlinearity, resulting in parameter reduction and parsimonious models. Note that the decoupled structure may be seen as a network, with $\boldsymbol{V}$ acting as weights, while allowing tailored nonlinear branches, contrary to the predefined activation functions typically found in neurons. This entails that the decoupled representation inherits all favourable properties of network models, e.g. neural networks.

The number of allowed univariate functions, denoted r, is crucial since it will determine whether the implied equivalence of (2) can be attained. A second linear transformation $\boldsymbol{W}$, maps the function back onto the outputs. The linear transformations then have the following dimensions: $\boldsymbol{V} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{W} \in \mathbb{R}^{n\times r}$.

Decoupling is a generic tool which is of practical use in, amongst others, the fields of nonlinear system identification and machine learning. The technique is designed to post-process multivariate functions which emerge naturally in data-driven modelling tasks.

[1] See section 2 for an overview of the notation used in this work.

1.1. Decoupling using first-order information

A corollary of (2) is that the Jacobians of the left- and the right-hand side need to match. This was exploited by [8] when proposing a decoupling algorithm. Denoting an evaluation of the Jacobian of the coupled function in point $\boldsymbol{p}(k)$ by $\boldsymbol{J}(k)$,

$$\boldsymbol{J}(k) \coloneqq \begin{bmatrix} \frac{\partial f_1}{\partial p_1}(k) & \dots & \frac{\partial f_1}{\partial p_m}(k) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial p_1}(k) & \dots & \frac{\partial f_n}{\partial p_m}(k) \end{bmatrix}, \tag{3}$$

and applying the chain rule in order to obtain the Jacobian of the decoupled structure, $\tilde{\boldsymbol{J}}(k)$ (again evaluated in $\boldsymbol{p}(k)$), we have that

$$\tilde{\boldsymbol{J}}(k) = \boldsymbol{W} \operatorname{diag}\left(\begin{bmatrix} g_1'(z_1(k)) & \cdots & g_r'(z_r(k)) \end{bmatrix}\right) \boldsymbol{V}^\top, \tag{4}$$

in which case $g_i'(z_i) \coloneqq \frac{\mathrm{d}g_i(z_i)}{\mathrm{d}z_i}$, and recalling that $\boldsymbol{z}(k) \coloneqq \boldsymbol{V}^\top \boldsymbol{p}(k)$. Dreesen et al. found that the decoupling task can be solved by a simultaneous diagonalisation of a set of Jacobian matrices [8]. The idea relies on the tri-linear form which arises when collecting evaluations of (4) in a third dimension. The process is depicted on the right-hand side of Fig. 1. Stacking N evaluations of $\tilde{\boldsymbol{J}}(k)$ in the third dimension expands the object into a three-way array $\tilde{\mathcal{J}} \in \mathbb{R}^{n \times m \times N}$, i.e. $\tilde{\mathcal{J}}[:,:,k] \coloneqq \tilde{\boldsymbol{J}}(k)$. Given the diagonal form (4), the collection of Jacobians may be written as a sum of r outer products (or rank-one terms). Element-wise we have that,

$$\tilde{\mathcal{J}}[o,\ell,k] = \sum_{i=1}^{r} \boldsymbol{W}[o,i]\boldsymbol{V}[\ell,i]\boldsymbol{G}'[k,i], \tag{5}$$

for $o = 1,\dots,n$ and $\ell = 1,\dots,m$ and $k = 1,\dots,N$. Here the matrix $\boldsymbol{G}' \in \mathbb{R}^{N \times r}$ stores the evaluations of the functions $g_i'(z_i)$, which appear along the central diagonal element of (4), i.e.

$$\boldsymbol{G}'[k,i] \coloneqq g_i'(z_i(k)). \tag{6}$$

A sum of rank-one terms defines a diagonal, or polyadic, tensor decomposition [9]. The bottom section of Fig. 1 depicts the diagonal core tensor which is obtained when extracting the diagonal plane, $\boldsymbol{G}'$[2]. It illustrates that the collection of Jacobians can be represented by three matrix factors, $\boldsymbol{W}$, $\boldsymbol{V}$, and $\boldsymbol{G}'$. The decomposed tensor may alternatively be written in shorthand notation

$$\tilde{\mathcal{J}} \coloneqq [\![\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}']\!]. \tag{7}$$

The structure of $\tilde{\mathcal{J}}$ is the key to solving the decoupling problem. It implies that a coupled function may be mapped to a decoupled counterpart by factoring a three-way tensor storing first order derivative information. This leads to the following algorithm:

1. Collect the Jacobian matrices of the known coupled function, $\boldsymbol{J}$, and stack them into a three-way array, i.e. the Jacobian tensor $\mathcal{J} \in \mathbb{R}^{n \times m \times N}$ (left-hand side of Fig. 1).

[2] In the original work of Dreesen et al. [8], and also in [6] $\boldsymbol{H}$ was used to denote $\boldsymbol{G}'$.

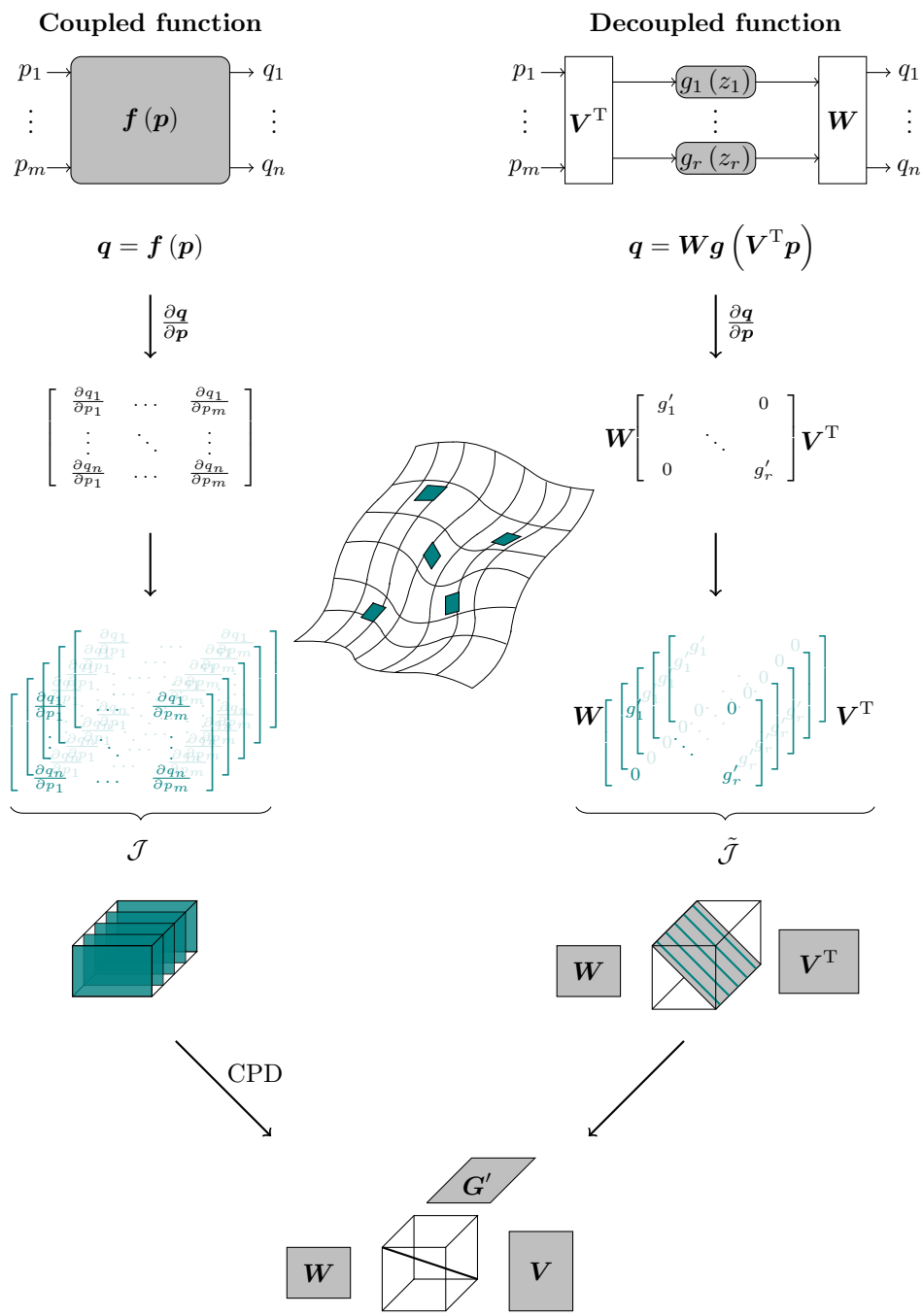


Figure 1: Overview of the decoupling procedure of [8] based on first order derivative information. The column on the left illustrates the construction of a third order Jacobian tensor $\mathcal{J}$ out of evaluations of the Jacobian of a coupled function along a number of operating points. The column on the right illustrates the corresponding steps applied to a decoupled function. Both functions are linked by computing the CPD of $\mathcal{J}$, i.e. an exact diagonal decomposition. If the CPD is unique, the approach retrieves an exact decoupled function. Observe that independent of the nature and complexity of $\boldsymbol{q} = \boldsymbol{f}(\boldsymbol{p})$, a third order tensor decomposition problem is retrieved, whereas this is not the case for the alternative methods described in [10].

2. Compute a diagonal (polyadic) tensor decomposition of $\mathcal{J}$, factoring it into $\{\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}'\}$. This leads to the approximation $\mathcal{J} \approx \tilde{\mathcal{J}}$ and thus $\mathcal{J} \approx [\![\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}']\!]$, where the accuracy of the approximation relies on r. In the original method of [8], the problem of exact decoupling was studied (referring to the equality in (2)). In that case also an exact tensor decomposition is required, hence the canonical polyadic decomposition (CPD) was used, which directly defines r (see section 2). In this work r will be considered a design parameter, opening up the possibility of approximate decoupling.

Recalling the definition of $\boldsymbol{G}'$ in (6), the columns $\boldsymbol{g}'_1, \ldots, \boldsymbol{g}'_r$ store collections of evaluations of the function derivatives $g'_i(z_i)$. This means that $\boldsymbol{G}'$ provides nonparametric estimates of $g'_i(z_i)$.

3. In a final step, the columns $\boldsymbol{g}'_i$ are parameterised as a function of z_i. A parametric estimates of $g_i(z_i)$ may then be obtained through integration.

The method of [8] is compelling for the following reason: irrespective of the nature of the coupled function, or the size of the function in terms of m and n, the procedure boils down to solving a third order tensor decomposition, whereas the complexity of alternative methods depend on the complexity of the original function, e.g. the degree of a multivariate polynomial [10]. It is, however, also subtle since it relies on two intrinsic assumptions:

1. The equality in (2), which implies that the decoupling is exact. Whether this is attainable is dictated by the number of branches, r. In Section 4 we will show that decoupled functions of the form of (2) are universal approximators.

2. The uniqueness of the tensor decomposition into its factors, $\{\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}'\}$, in order for the elements of $\boldsymbol{g}'_i$ to truly correspond to evaluations of $g'_i(z_i)$. Whether the decomposition is unique again relies on r [11].

It turns out that the requirements on r for unique decompositions and accurate decoupled structures can be conflicting, imposing strong limitations on the applicability of the method. Fig. 2 illustrates a potential solution of $\boldsymbol{G}'$ in the case of a non-unique decomposition. The figure corresponds to the toy problem of Section 7.1 in which a third order polynomial, expressed in the standard monomial basis, is decoupled into three univariate branches. Although an exact solution exists, it is clear that the decomposition does not convey information on the univariate functions. In this work, a new method is presented, leveraging the powerful idea of decomposing the Jacobian tensor while tackling the challenges of approximate decoupling and non-unique decompositions. The proposed method enables to retrieve smooth estimates of decoupled functions in a nonparametric way.

2. Notation

Vectors are denoted by lower-case bold-faced letters, e.g. $\boldsymbol{a} \in \mathbb{R}^p$. Matrices are denoted by bold-faced upper-case letters, e.g. $\boldsymbol{A}$. The columns of a matrix are denoted by bold-faced lower-case letters, e.g. we have $\boldsymbol{A} = [\boldsymbol{a}_1 \quad \ldots \quad \boldsymbol{a}_r] \in \mathbb{R}^{p \times r}$. Tensors are denoted by calligraphic letters, e.g. $\mathcal{A} \in \mathbb{R}^{p \times r \times N}$. Matrix slices along the third dimension of a third order tensor are denoted by subscript indices, e.g. $\boldsymbol{A}_i$, alternatively square brackets are used, i.e. $\boldsymbol{A}_i \coloneqq \mathcal{A}[:, :, i]$. Scalars are given lower case letters. The element of a vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$ or a (third-order) tensor $\mathcal{A}$ may hence be denoted as a_i, a_{ij} or a_{ijk}, respectively. Alternatively square brackets are used, e.g. $a_{ij} \coloneqq \boldsymbol{A}[i, j]$.

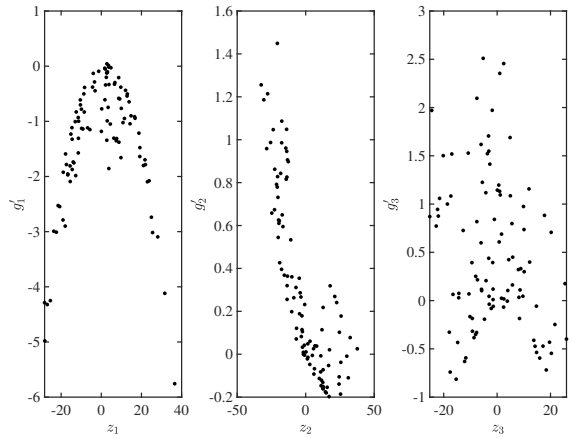


Figure 2: Visualisation of the nonparametric estimates of the functions $g_i'(z_i), i = 1, \ldots, r$, which are obtained as the columns of matrix factor $\boldsymbol{G}' = \left[\boldsymbol{g}_1', \ldots, \boldsymbol{g}_r'\right]$, see 6 for a definition of $\boldsymbol{G}'$. The result corresponds to the toy problem of Section 7.1 in which a third order polynomial, expressed in the standard monomial basis, is decoupled into three univariate branches. Although an exact solution exists, the decomposition is unable to convey information on the univariate functions, given that it is non-unique. Non-uniqueness of the tensor decomposition has been a limiting factor to the applicability of the method proposed by [8]. This limitation is overcome by the methods presented in this work.

A collection of N unsorted elements, e.g. vectors $\boldsymbol{a}$, is denoted by $\{\boldsymbol{a}(k)\}_{k=1}^{N}$. The root mean square of a vector is denoted by $\text{rms}(\boldsymbol{a}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} a_i^2}$. The expected value is indicated by $\mathbb{E}$.

The Kronecker product of two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is denoted $\boldsymbol{A} \otimes \boldsymbol{B}$. The Khatri-Rao product of two matrices is denoted by $\boldsymbol{A} \odot \boldsymbol{B}$ [9]. The Hadamard (element-wise) product of $\boldsymbol{A}$ and $\boldsymbol{B}$ is denoted by $\boldsymbol{A} * \boldsymbol{B}$. The pseudo-inverse of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^{\dagger}$. The vectorisation operator that converts a matrix or a tensor to a vector by concatenating its columns is denoted by $\text{vec}(\boldsymbol{A}) = \boldsymbol{a}$. The mode-$n$ unfolding of a tensor $\mathcal{A}$ is denoted by $\boldsymbol{A}_{(n)}$. The columns of $\boldsymbol{A}_{(n)}$ are the mode-n fibers (columns, rows, tubes) of $\mathcal{A}$ [9]. CPD is used to denote the canonical polyadic decomposition of a tensor, i.e. a sum of the minimal number of rank-one terms for which the decomposition is exact. FTD is used to denote a filtered tensor decomposition, referring to the procedure which is developed in this work.

3. Problem statement

The objective is to have a procedure which allows one to approximate any arbitrary continuous multivariate function $\boldsymbol{f}(\boldsymbol{p})$ by a decoupled function. In practice $\boldsymbol{f}(\boldsymbol{p})$ is typically inferred from data during a training process. It is therefore justified to define the function approximation on the basis of N operating points, drawn from the training distribution. This will ensure that the approximation is accurate in the domain of interest. The objective then assumes the form

$$\underset{\boldsymbol{W},\boldsymbol{V},\boldsymbol{g}\in\mathcal{G}}{\arg\min} \frac{1}{N}\sum_{k=1}^{N} \|\boldsymbol{f}(\boldsymbol{p}(k)) - \boldsymbol{W}\boldsymbol{g}(\boldsymbol{V}^{\top}\boldsymbol{p}(k))\|_2^2, \tag{8}$$

with $\mathcal{G}$ a predefined function space. A direct approach to solving (8) would entail solving a hard nonlinear optimisation problem. Instead, and similar to [8], we will rely on the connection to tensor decompositions, observed when formulating the problem at the Jacobian level,

$$\underset{\boldsymbol{W},\boldsymbol{V},\boldsymbol{G}'}{\arg\min} \|\mathcal{J} - [\![\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}']\!]\|_F^2. \tag{9}$$

Replacing (8) by (9) is not without consequence. Both the motivation and implications are listed below:

1. The main motivation is that (9) is tri-linear in the unknown factors. This opens up the problem to an alternating approach where by iteratively fixing two of the factors, the remaining problem of optimising the third factor admits a closed-form solution.

2. (9) is not constrained by a function family (this would destroy the multi-linearity). In other words, (6) is not explicitly enforced. Instead, [8] relies on the uniqueness property of the tensor decomposition. In case of a unique decomposition, the factor $\boldsymbol{G}'$ will indeed store nonparametric estimates of the functions $g_i'(z_i)$, as given by (6).

3. Formulating the function approximation at the Jacobian level implies that the zeroth-order information is lost. An additional step will be required in order to recover the constant terms. This will addressed in section 6.

4. The solution obtained from (9) is not necessarily optimal with respect to (8). In such cases a post-optimisation based on (9) can be used, where the solution obtained from the decoupling procedure serves as starting values.

The objective of this work is to have a procedure that no longer relies on the uniqueness property of the decomposition, which was found to be a limiting factor for the applicability of function decoupling. In order to be able to let go of a unique solution and still retrieve meaningful decoupled functions it is proposed to promote smoothness of the nonparametric function estimates. Promoting smoothness will prevent that non-meaningful solutions, similar to Fig. 2 are obtained. The benefit is twofold:

- By removing the requirement for uniqueness, r can become a design parameter. Or in other words, the complexity of the decoupled target function can be tuned and balanced with respect to the accuracy of the approximation.

- Promoting smoothness leads to meaningful results without having to constrain the function family. The proposed decoupling procedure therefore remains nonparametric, allowing any type of function to be handled.

Smoothness is, however, not an objective that is to be quantified in itself. The level of required smoothness goes hand in hand with the final choice of basis expansion used to parametrise the branches, e.g. the degree of the polynomial basis. What will be quantified and monitored is the accuracy of the function approximation, for which a metric is introduced in (42), i.e. a normalised version of (8).

3.1. Contributions presented in this work

Two new algorithms will be introduced. Both methods are distinct in how smoothness is achieved. In Section 5.2 smoothness is encoded implicitly in the objective. The result is a fully automated procedure requiring very little user interaction. In Section 5.3 regularisation is used to explicitly promote continuity of the derivative. An illustration will show that it is very intuitive to penalise non-smooth solutions.

Alternative methods that aim at ensuring unique decompositions rely either on introducing information of the Hessian, or on introducing explicit polynomial constraints on the unknown g_i [12, 13]. Note that such constraints do discriminate function families from the start. In this work a first-order approach is applied, avoiding the prohibitively expensive computation of the Hessian, while resorting to nonparametric methods to ensure smoothness, hence keeping the decomposition independent of any choice of basis expansion.

4. Universal approximation theorem

It may be questioned under what conditions decoupled functions of the form of (2) can be used to approximate an arbitrary continuous multivariate function. In what follows we study the universal approximation properties of such decoupled forms.

Theorem 1. *Let A be the family of real continuous functions, defined on a compact set $K \subset \mathbb{R}^n$ that assume the form*

$$h_r(\boldsymbol{p}) = \sum_{i=1}^{r} w_i g_i(\boldsymbol{v}_i^\top \boldsymbol{p}), \tag{10}$$

for some integer $r > 0$, coefficients w_i, and vectors $\boldsymbol{v}_i \in \mathbb{R}^n$. Here, g_i is a collection of univariate functions $g_i : \mathbb{R} \mapsto \mathbb{R}$. Let C denote the space of all real continuous functions on K. There exists a collection of univariate functions g_i such that A is dense in C under the infinity or amplitude norm. That is, for any $f \in C$ and any $\epsilon > 0$, there exists $r > 0$, vectors $\boldsymbol{v}_i$ and coefficients w_i such that $h_r \in A$ satisfies

$$\|f - h_r\|_\infty \coloneqq \sup_{\boldsymbol{p} \in K} |f(\boldsymbol{p}) - h_r(\boldsymbol{p})| < \epsilon. \tag{11}$$

Proof. Consider the family of monomial interpolation functions $g_i(x) \coloneqq x^i$. The corresponding functions $h_r^\star \in A$ assume the form

$$h_r^\star(\boldsymbol{p}) = \sum_{i=1}^{r} w_i [\boldsymbol{v}_i^\top \boldsymbol{p}]^i. \tag{12}$$

Given that the monomials $[\boldsymbol{v}_i^\top \boldsymbol{p}]^i$ separate points of K and are closed under multiplication, finite linear combinations of such functions form a point separating algebra which is dense in C under the sup norm because of the Stone-Weierstrass theorem [14]. □

Remark. Note that the choice of interpolation function is not restricted to monomials, e.g. the family of exponential functions $\sum_{i=1}^{r} w_i e^{\boldsymbol{v}_i^\top \boldsymbol{p}}$, with w_i real, and $\boldsymbol{v}_i$ a vector of non-negative integers, would also do [15]. Moreover, observe that single-layer neural networks are a subset of the decoupled form. Therefore, the classically considered activation functions may also be considered [2].

5. Nonparametric Filtered Tensor Decomposition (FTD)

The proposed idea to ensure smoothness revolves around introducing *finite difference filters* into the decomposition routine of the Jacobian tensor. A finite difference filter is a matrix which upon multiplication with a vector of function evaluations, returns a finite difference approximation. The use of finite differences allows one to steer the decomposition towards meaningful results in a nonparametric way.

Consider the couple of N evaluations of the function $g_i(z_i)$, i.e. $\{z_i(k), g_i(k)\}_{k=1}^{N}$. Given that $g_i'(z_i) \coloneqq \frac{\mathrm{d}g_i(z_i)}{\mathrm{d}z_i}$, we can compute a finite difference approximation, $\boldsymbol{g}_i' \in \mathbb{R}^N$, by applying a finite difference filter to the vector of evaluations $\boldsymbol{g}_i$,

$$\boldsymbol{g}_i' = \boldsymbol{S}_i^{-1}\boldsymbol{D}_i\boldsymbol{S}_i\boldsymbol{g}_i, \tag{13}$$

where $\boldsymbol{S}_i$ is a sorting matrix, arranging the elements of $\boldsymbol{g}_i$ in ascending order of z_i, and $\boldsymbol{D}_i$ embodies the finite difference operator. Should $\boldsymbol{z}_i$ be equidistantly spaced, with a spacing of δ_{z_i}, $\boldsymbol{D}_i$ could for instance be defined by the Toeplitz matrix

$$\boldsymbol{D}_i = \frac{1}{\delta_{z_i}} \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 1 \end{bmatrix}, \tag{14}$$

where this particular example is known as a 2-points forward differencing scheme (or a right-derivative approximation). In this context, the term *finite difference filter*, is used to refer to the sequence of operations

$$\boldsymbol{F} \coloneqq \boldsymbol{S}^{-1}\boldsymbol{D}\boldsymbol{S}. \tag{15}$$

In practice, $\boldsymbol{z}_i$, will not be equidistantly spaced. A generalisation of $\boldsymbol{D}$ is provided in Appendix A. Introducing finite difference filters will prove beneficial in two ways:

- It will allow $\mathcal{J}$ to be decomposed into $\{\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}\}$, where $\boldsymbol{G}$ stores evaluations of the functions g_i, hence removing the need for an integration step (Section 5.1).
- It will provide the means to include a smoothness objective in the decomposition (Sections 5.2 and 5.3) [16, 17].

5.1. A decomposition into $\boldsymbol{W}$,$\boldsymbol{V}$, and $\boldsymbol{G}$

Exploiting the relationship of (13) we may propose an alternative factorisation of the Jacobian tensor where the third factor contains $\boldsymbol{G}$, i.e a matrix holding evaluations of the to be determined univariate functions $g_i(z_i)$ along its columns. This is in contrast to the factor $\boldsymbol{G}'$ which stores derivative information. The sought-after decomposition is then of the form

$$\mathcal{J} \approx [\![\boldsymbol{W}, \boldsymbol{V}, \mathcal{F}(\boldsymbol{V}) \circ \boldsymbol{G}]\!], \text{ with} \tag{16}$$

$$\boldsymbol{G}' \coloneqq \mathcal{F}(\boldsymbol{V}) \circ \boldsymbol{G}. \tag{17}$$

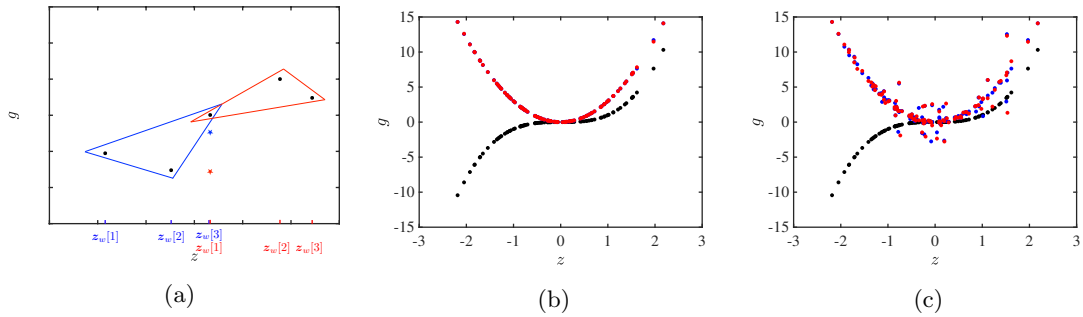


Figure 3: (a) Black markers indicate a set of non-smooth evaluations $g(z)$. The blue and red $\star$ respectively denote the left and right 3-points finite difference, computed on the left and right window of points. (b) Black markers are evaluations of the smooth function $g = z^3$ for a set of randomly chosen points $\boldsymbol{z}$. Blue and red (almost on top of each other) are respectively the left and right 3-points finite differences. (c) Distorting the evaluations with Gaussian noise ($\sigma = 0.001\,\mathrm{rms}(\boldsymbol{g})$) results in clear differences amongst both finite difference approximations.

The operation '$\circ$' symbolises the column-wise filtering of (13)[3]. For convenience, the finite difference filters, corresponding to each column, are stored in a three-way array $\mathcal{F} \in \mathbb{R}^{N\times N\times r}$ such that

$$\mathcal{F}[:,:,i](\boldsymbol{V}) \coloneqq \boldsymbol{F}_i(\boldsymbol{V}) = \boldsymbol{S}_i^{-1}\boldsymbol{D}_i\boldsymbol{S}_i. \tag{18}$$

Notice the explicit dependence on the factor $\boldsymbol{V}$. The factor $\boldsymbol{V}$ defines the axes on the basis of which the finite difference is computed. Both the sorting matrix $\boldsymbol{S}_i$ and the finite difference operator $\boldsymbol{D}_i$ therefore depend on $\boldsymbol{V}$ (see Appendix A). (17) may alternatively be written in vectorised form

$$\mathrm{vec}\left(\boldsymbol{G}'\right) = \begin{bmatrix} \boldsymbol{F}_1 & 0 & \dots & 0 \\ 0 & \boldsymbol{F}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{F}_r \end{bmatrix} \mathrm{vec}\left(\boldsymbol{G}\right). \tag{19}$$

Given that the original decomposition $\{\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}'\}$ has been translated to $\{\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}\}$, also the smoothness requirement, which was formulated in Section 3, is transferred onto $\boldsymbol{G}$.

5.2. Adding an implicit smoothness objective

Smoothness and finite differences are intimately linked, e.g. abrupt local changes will be emphasised by a finite difference filter. In Appendix A a procedure to construct multiple finite difference filters which locally act on a different window of points, was introduced. Given that all these filters all approximate the first order derivative, differences amongst their results convey information on the local smoothness of the function.

Fig. 3a illustrates the operation of two distinct finite difference filters on a set of non-equidistantly spaced points. Both filters are created in accordance to Appendix A. In blue, a *3-points left finite difference* is used. The result is indicated with a blue $\star$. In red, a *3-points right finite difference*

[3]This notation should not be confused with that of the outer product.

is used (red $\star$). Both approximations yield a different result. This is due to local non-smoothness. Fig. 3b and 3c demonstrate this property in an intuitive manner. Fig. 3b depicts a number of evaluations of the smooth function $g = z^3$. In Fig. 3c, these evaluations are distorted by Gaussian noise with $\sigma = 0.001\,\mathrm{rms}(\boldsymbol{g})$. In both cases a 3-points left and a 3-points right finite difference is computed. The results are indicated by blue and red markers, respectively. While for the smooth function hardly any difference may be observed, the distorted evaluations clearly result in distinct derivative approximations. This implies that the deviation between left and right finite difference filters conveys information on the local smoothness of the evaluations. This property will be exploited in order to promote smooth solutions.

The example in Fig. 3 alludes to the fact that decompositions of the form of (16), which simultaneously satisfy multiple finite difference filters, must be smooth. In other words, considering multiple filters limits the number of possible solutions to a subset of smooth solutions. Smoothness can hence implicitly be encoded in the decomposition by constructing objective functions, for each matrix factor, on the basis of multiple filters.

Consider s collections of finite difference filters, stored in the three-way arrays $\mathcal{F}_1$ to $\mathcal{F}_s$, each constructed on the basis of a different window of points (see Appendix A). The claim is that only smooth solutions of $\boldsymbol{G}$ may satisfy

$$\mathcal{J} \approx [\![\boldsymbol{W}, \boldsymbol{V}, \mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G}]\!] \approx \ldots \approx [\![\boldsymbol{W}, \boldsymbol{V}, \mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G}]\!]. \tag{20}$$

In what follows we will introduce an algorithm able to solve the decomposition expressed by (20).

5.2.1. Implicit algorithm

A classical approach to solving a tensor decomposition is by tensor unfoldings in combination with an alternating least-squares (ALS) procedure. Denoting the unfolding of the third order tensor $\mathcal{J}$ along its rows, columns and tubes by $\boldsymbol{J}_{(1)}, \boldsymbol{J}_{(2)}, \boldsymbol{J}_{(3)}$, respectively, and using '$\odot$' to denote the Khatri-Rao product, we may write [9]

$$\boldsymbol{J}_{(1)} \approx \boldsymbol{W}((\mathcal{F}_i(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V})^\top, \tag{21a}$$

$$\boldsymbol{J}_{(2)} \approx \boldsymbol{V}((\mathcal{F}_i(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{W})^\top, \tag{21b}$$

$$\boldsymbol{J}_{(3)} \approx (\mathcal{F}_i(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top, \tag{21c}$$

with $i = 1, \ldots, f$. In the first step of the procedure the factors $\{\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}\}$ are initialised with random numbers. Following the ALS approach [18, 19], each matrix factor is then updated while treating both others as constants. This leads to the following objective functions and update formulas:

1. Update $\boldsymbol{W} \to \boldsymbol{W}^+$

   From combining (20) and (21a), the following objective is inferred

   $$\arg\min_{\boldsymbol{W}} \|\boldsymbol{J}_{(1)} - \boldsymbol{W}((\mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V})^\top\|_F^2 + \ldots + \|\boldsymbol{J}_{(1)} - \boldsymbol{W}((\mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V})^\top\|_F^2. \tag{22}$$

   Since $\boldsymbol{W}$ appears linearly in the objective, an analytical update formula is obtained (Appendix B.1).

2. Update $\boldsymbol{V} \to \boldsymbol{V}^+$

Combining (20) and (21b) leads to

$$\arg\min_{\boldsymbol{V}} \|\boldsymbol{J}_{(2)} - \boldsymbol{V}((\mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{W})^\top\|_F^2 + \ldots + \|\boldsymbol{J}_{(2)} - \boldsymbol{V}((\mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{W})^\top\|_F^2. \quad (23)$$

Given that $\boldsymbol{V}$ appears nonlinearly in the objective, nonlinear optimisation is required to compute an update $\boldsymbol{V}^+$. The optimisation is solved using a built-in MATLAB Levenberg-Marquardt solver [20]. As initialisation a relaxation, based on one of the filters is used. More specifically, the dependency of $\mathcal{F}_i$ on $\boldsymbol{V}$ is disregarded. Denoting $\boldsymbol{G}_i' \coloneqq \mathcal{F}_i \circ \boldsymbol{G}$, we have that

$$\boldsymbol{V}_0^+ = \boldsymbol{J}_{(2)} \left(\left(\boldsymbol{G}_i' \odot \boldsymbol{W} \right)^\top \right)^\dagger. \quad (24)$$

Using the property

$$(\boldsymbol{A} \odot \boldsymbol{B})^\top (\boldsymbol{A} \odot \boldsymbol{B}) = \left(\boldsymbol{A}^\top \boldsymbol{A} \right) * \left(\boldsymbol{B}^\top \boldsymbol{B} \right), \quad (25)$$

with '$*$' denoting the Hadamard product, a more efficient update formula may obtained, requiring only the inversion of an $r \times r$ matrix. The initialisation of the update then becomes

$$\boldsymbol{V}_0^+ = \boldsymbol{J}_{(2)} \left(\boldsymbol{G}_i' \odot \boldsymbol{W} \right) \left(\left(\boldsymbol{G}_i'^\top \boldsymbol{G}_i' \right) * \left(\boldsymbol{W}^\top \boldsymbol{W} \right) \right)^\dagger. \quad (26)$$

To improve conditioning every update is followed by a normalisation such that the columns are given by $\frac{\boldsymbol{v}_i^+}{\|\boldsymbol{v}_i^+\|_2}$.

3. Update $\boldsymbol{G} \to \boldsymbol{G}^+$

   In an analogous way $\boldsymbol{G}$ is updated from combining (20) and (21c),

$$\arg\min_{\boldsymbol{G}} \|\boldsymbol{J}_{(3)} - (\mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top\|_F^2 + \ldots + \|\boldsymbol{J}_{(3)} - (\mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top\|_F^2. \quad (27)$$

   It can be shown that $\boldsymbol{G}$ appears linearly in the objective, leading to an analytical update formula (Appendix B.1).

Steps 1 to 3 are iterated until convergence or a stopping criterium is met. The updated factors are passed on from one iteration to the next. The univariate functions $g_i(z_i)$ are ultimately obtained from parameterising the columns $\boldsymbol{g}_i$ using a basis expansion which can be freely chosen by the user.

The *implicit* smoothness procedure results in a robust method for decoupling continuous multivariate functions, regardless of the function family. Moreover, only very little user interaction is needed. The general procedure allows one to include as many filters as required. In Section 7.1 we will show that, in practice, including two filters, e.g. a left and a right derivative filter, may be enough to ensure continuity of the derivative and hence smoothness of the solution. Additionally, the smoothness property may be relaxed by constructing filters which allow a finite number of discontinuities. Loosely said, the filters may be seen as moulds, shaping the functions into a desired form. One may for instance also envisage encoding saturating behaviour into the filters.

5.3. Adding an explicit smoothness objective

In the previous section, the solutions of the decomposition could be brought back to a smooth subset by requiring that the solution satisfies multiple filters. In an analogue way, multiple filters may also be used to explicitly express a smoothness objective.

Consider the following collections of filters: a collection of 3-points left finite difference filters, denoted by $\mathcal{F}_L$, 3-points right filters, denoted by $\mathcal{F}_R$ and 3-points central filters, denoted by $\mathcal{F}_C$. In analogy with (16), the decomposition is of the form

$$\mathcal{J} \approx [\![\boldsymbol{W}, \boldsymbol{V}, \mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G}]\!]. \tag{28}$$

Using the standard unfoldings, the decomposition can be translated to the following equivalent expressions [9]

$$\boldsymbol{J}_{(1)} \approx \boldsymbol{W}((\mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V})^\top, \tag{29a}$$

$$\boldsymbol{J}_{(2)} \approx \boldsymbol{V}((\mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{W})^\top, \tag{29b}$$

$$\boldsymbol{J}_{(3)} \approx (\mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top. \tag{29c}$$

When inferring objective functions from (29a) up to (29c), appropriate penalty terms will be added using regularisation. In this work we will make use of regularisation to penalise divergent results from a left and a right finite difference filter. Minimising the difference between the left and the right derivative favours differentiable functions (and therefore also continuity). This will steer the decomposition towards smooth solutions. It must be stressed that other penalty terms, reflecting possibly other desired properties, may also be included.

5.3.1. Explicit algorithm

The factors $\{\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}\}$ are initialised with random numbers. Also here an ALS approach is adopted. This leads to the following objective functions and update formulas:

1. Update $\boldsymbol{W} \to \boldsymbol{W}^+$

   From Eq.(29a) we infer the following objective

$$\arg\min_{\boldsymbol{W}} \|\boldsymbol{J}_{(1)} - \boldsymbol{W}((\mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V})^\top\|_F^2. \tag{30}$$

   For convenience we will denote $\boldsymbol{G}'_C \coloneqq \mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G}$. Given that $\boldsymbol{W}$ appears linearly in the objective, an analytical update formula is obtained,

$$\boldsymbol{W}^+ = \boldsymbol{J}_{(1)} \left(\left(\boldsymbol{G}'_C \odot \boldsymbol{V} \right)^\top \right)^\dagger. \tag{31}$$

   Using the property

$$(\boldsymbol{A} \odot \boldsymbol{B})^\top (\boldsymbol{A} \odot \boldsymbol{B}) = \left(\boldsymbol{A}^\top \boldsymbol{A} \right) * \left(\boldsymbol{B}^\top \boldsymbol{B} \right), \tag{32}$$

   leads to the more efficient update formula, requiring only the inversion of an $r \times r$ matrix

$$\boldsymbol{W}^+ = \boldsymbol{J}_{(1)} \left(\boldsymbol{G}'_C \odot \boldsymbol{V} \right) \left(\left(\boldsymbol{G}'^\top_C \boldsymbol{G}'_C \right) * \left(\boldsymbol{V}^\top \boldsymbol{V} \right) \right)^\dagger. \tag{33}$$

   Notice that the smoothness of $\boldsymbol{G}$ does not depend on $\boldsymbol{W}$.

2. Update $\boldsymbol{V} \to \boldsymbol{V}^+$

   Apart from the tensor approximation objective, which is implied by (29b), the update of $\boldsymbol{V}$ must also take into account the smoothness of $\boldsymbol{G}$. The smoothness is indeed affected by $\boldsymbol{V}$

since $\boldsymbol{V}$ defines the z-axis, $\boldsymbol{z} = \boldsymbol{V}^\top \boldsymbol{p}$ (see Fig. 2). The smoothness objective is therefore explicitly added using regularisation,

$$\arg\min_{\boldsymbol{V}} \|\boldsymbol{J}_{(2)} - \boldsymbol{V}((\mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{W})^\top\|_F^2 + \lambda \| (\mathcal{F}_L(\boldsymbol{V}) \circ \boldsymbol{G}) - (\mathcal{F}_R(\boldsymbol{V}) \circ \boldsymbol{G}) \|_F^2, \tag{34}$$

where λ is a hyperparameter which balances both objectives. The additional term penalises divergent results from a left and a right finite difference filtering operation, ultimately steering the optimisation towards smooth solutions. The penalty term in (34) was preferred over classical Tikhonov regularisation [21], i.e. a penalty of the form $\|\mathcal{F}_C \circ \boldsymbol{G}\|_F^2$, which was found to have the tendency to overly flatten the function estimates.

Given that $\boldsymbol{V}$ appears nonlinearly in (34), nonlinear optimisation is required to obtain $\boldsymbol{V}^+$. In order to promote smoothness evenly on all the columns of $\boldsymbol{G}$, a normalisation is needed. Denoting $\boldsymbol{G}'_L \coloneqq \mathcal{F}_L \circ \boldsymbol{G}$, and $\boldsymbol{G}'_R \coloneqq \mathcal{F}_R \circ \boldsymbol{G}$, all columns are normalised by their rms value, i.e. $\boldsymbol{g}'_{L_i}/\mathrm{rms}(\boldsymbol{g}'_{L_i})$, and $\boldsymbol{g}'_{R_i}/\mathrm{rms}(\boldsymbol{g}'_{R_i})$.

The optimisation is solved using a built-in MATLAB Levenberg-Marquardt solver [20]. For an analytical approach to the optimisation problem, the reader is referred to [22]. As initialisation a relaxation is used (i.e. without penalty term and disregarding the dependency of $\mathcal{F}_C$ on $\boldsymbol{V}$)

$$\boldsymbol{V}_0^+ = \boldsymbol{J}_{(2)} \left(\boldsymbol{G}'_C \odot \boldsymbol{W}\right) \left(\left({\boldsymbol{G}'_C}^\top \boldsymbol{G}'_C\right) * \left(\boldsymbol{W}^\top \boldsymbol{W}\right)\right)^\dagger. \tag{35}$$

To improve conditioning every update is followed by a normalisation such that the columns are given by $\frac{\boldsymbol{v}_i^+}{\|\boldsymbol{v}_i^+\|_2}$.

3. Update $\boldsymbol{G} \to \boldsymbol{G}^+$

   In analogy with (34), the update formula of $\boldsymbol{G}$ is also found from a joint objective function.

$$\arg\min_{\boldsymbol{G}} \|\boldsymbol{J}_{(3)} - (\mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top\|_F^2 + \lambda \| (\mathcal{F}_L(\boldsymbol{V}) \circ \boldsymbol{G}) - (\mathcal{F}_R(\boldsymbol{V}) \circ \boldsymbol{G}) \|_F^2. \tag{36}$$

   It can be shown that $\boldsymbol{G}$ appears linearly in (36). This allows for an analytical update formula (given in Appendix B.2).

Steps 1 to 3 are iterated until convergence is reached or a stopping criterium is met. The updated factors are passed on from one iteration to the next. The appropriate value of λ is to be determined from a line search. The univariate functions $g_i(z_i)$ are ultimately obtained from parameterising the columns $\boldsymbol{g}_i$ using a basis expansion which can be freely chosen by the user.

Explicitly penalising divergent results from left and right finite difference filters is an intuitive way of introducing smoothness into the objective. An illustration in Section 7.1 will demonstrate that the level of smoothness responds as anticipated to the choice of the hyperparameter λ. It allows for a controlled balancing between accuracy of the decomposition and smoothness. The added flexibility comes at the price of increased computing times when scanning over λ.

6. Estimating the constant terms

Inherent to decoupling on the basis of first-order derivative information is the loss of the constant terms. The constant term of every function f_i of $\boldsymbol{f}(\boldsymbol{p}) : \mathbb{R}^m \mapsto \mathbb{R}^n$ may, however, be directly

estimated from input-output data. Let $\Delta_f \in \mathbb{R}^{N\times n}$ denote the error between $\boldsymbol{f}$ and the retrieved decoupled function $\boldsymbol{f}_D$, computed on the operating points such that

$$\Delta_f[i,j] = f_j\left(\boldsymbol{p}(i)\right) - f_{D_j}\left(\boldsymbol{p}(i)\right), \tag{37}$$

with $j = 1,\ldots,n$ and $i = 1,\ldots,N$. A least-squares estimate of the constant terms, $\boldsymbol{c} \in \mathbb{R}^n$, is then found from

$$\boldsymbol{c}^\top = \left(\mathbf{1}_N^\top \mathbf{1}_N\right)^{-1} \mathbf{1}_N^\top \Delta_f, \tag{38}$$

with $\mathbf{1}_N \in \mathbb{R}^N$ a vector of ones.

The constant terms may be incorporated in $\boldsymbol{f}_D = \boldsymbol{W}\boldsymbol{g}\left(\boldsymbol{V}^\top \boldsymbol{p}\right)$ by adding an additional branch

$$\tilde{\boldsymbol{f}}_D = \begin{bmatrix}\boldsymbol{W} & \boldsymbol{c}\end{bmatrix} \begin{bmatrix} g_1(z_1) \\ \vdots \\ g_r(z_r) \\ 1 \end{bmatrix}. \tag{39}$$

7. Case study 1: a polynomial example

The objective is to demonstrate the capability of the method to return decoupled functions irrespective of the tensor decomposition properties (i.e. for non-exact and non-unique decompositions). The first example is of the polynomial type. In Section 8 a neural network will be studied. Recall that, given the universal approximation theorem of Section 4, any continuous function may be approximated by a decoupled form.

7.1. Toy problem

Both the explicit procedure of Section 5.3 and the implicit method of Section 5.2 will first be demonstrated on a toy problem. Consider the decoupled polynomial function

$$\boldsymbol{f}_D(\boldsymbol{p}) = \overbrace{\begin{bmatrix} 3 & 0.5 & -1 \\ 1 & 2 & 3 \end{bmatrix}}^{\boldsymbol{W}} \overbrace{\begin{bmatrix} z_1^3 + 0.5 z_1^2 \\ 2z_2^3 + z_2^2 \\ z_3^3 + 3z_3^2 \end{bmatrix}}^{\boldsymbol{g}(\boldsymbol{z})} \tag{40a}$$

$$\boldsymbol{z} = \overbrace{\begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 0.5 & 3 \end{bmatrix}}^{\boldsymbol{V}^\top} \boldsymbol{p}, \tag{40b}$$

for which $m = 2$, $n = 2$ and $r = 3$. An equivalent[4] formulation, expressed in the standard monomial basis reads

$$\boldsymbol{f}(\boldsymbol{p}) = \begin{bmatrix} 5.25 & 0 & -20.5 & 29.875 & 42.75 & 31.5 & -2 \\ 20.75 & 41 & 85 & 109.375 & 120.75 & 88.5 & 93 \end{bmatrix} \begin{bmatrix} p_1^2 \\ p_1 p_2 \\ p_2^2 \\ p_1^3 \\ p_1^2 p_2 \\ p_1 p_2^2 \\ p_2^3 \end{bmatrix}. \tag{41}$$

One may think of (41) as a standalone static multivariate polynomial although it might as well be the internal part of a dynamical model. It could for instance be describing a polynomial NARX model [23], in which case p_1 and p_2 would represent (delayed) input or output samples and n would be equal to 1. Furthermore, it could be the nonlinear function which describes the state update in nonlinear state-space models, in which case p_1 and p_2 would be either state or input variables [6, 7].

The objective is to retrieve the decoupled function of the form (40) starting from (41).

1. The first step consists in evaluating the Jacobian of (41), $\boldsymbol{J}$, and stacking the matrices in the third dimension. A number of $N = 100$ operating points were randomly selected from $\mathcal{U}(-1.5, 1.5)$, leading to $\mathcal{J} \in \mathbb{R}^{2\times 2\times 100}$. In practice the operating points should cover the domain of interest.

2. The next step involves decomposing the Jacobian tensor. At this stage the number of univariate branches that make up the decoupled function, are selected. By construction we know that for the given example a solution containing $r = 3$ branches exists ((40)).

   (a) **CPD**: First we examine the classical canonical polyadic decomposition (CPD), which returns $\mathcal{J} \approx [\![\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{G}']\!]$. With a tensor approximation ((9)) close to machine precision, the decomposition was successful. Studying the result, however, reveals an underlying problem (depicted in Fig. 2). As may be observed, the decomposition did not convey information on the univariate functions, which make up (40). The decomposition only points to (40) in case it is unique, which is not the case in this example for $r = 3$.

   (b) **FTD implicit**: Not all solutions of the decomposition prove useful in our search for the decoupled function. We can, however, limit the possible solutions to a smooth subset by using an objective based on multiple finite difference filters, i.e. using the implicit algorithm of Section 5.2. Given that in this case, r is a design parameter, we can study the solution for a range of r irrespective of whether an exact decoupled form exists. We will construct the objective ((20)) on the basis of two filters, a 3-points left finite difference and a 3-points right finite difference. The procedure is then fully automated. The results for a scan of $r = 1$ to 4 are presented in Fig. 4.

   (c) **FTD explicit**: Using the explicit algorithm of Section 5.3 we can steer the decomposition towards appropriate solutions by penalising non-smooth behaviour. Also in this case r is a design parameter. Fig. 5 depicts the role of the hyperparameter λ for decompositions with $r = 3$.

[4] The equivalency is subject to rounding errors in the computation of the coefficients.

3. To finalise the decoupling procedure, the estimates of the functions g_i are parameterised. At this stage any basis expansion which deems fit can be used, provided that it is linear in the parameters. In Fig. 4 and 5b accurate third order polynomials were fitted (red curves). To assess the accuracy of the decoupled function, a relative root-mean-squared error is computed on the operating points

$$e_i = \frac{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(f_i(\boldsymbol{p}(k) - f_{D_i}(\boldsymbol{p}(k))\right)^2}}{\sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(f_i(\boldsymbol{p}(k)) - \mathbb{E}\left(f_i\right)\right)^2}} \times 100, \tag{42}$$

with $\mathbb{E}\left(f_i\right)$ the expected value over all N points. The results are listed in Table 1. In case of the explicit method, the lowest error found from a coarse search over $\sqrt{\lambda} = 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}$, is reported.

Accurate decoupled approximations of (41) were retrieved. Note that r can be used to balance model complexity to accuracy. The exact decoupled function, (40) is, however, not retrieved given that the ALS algorithm leads to a local minimum of the objectives. Adding additional flexibility, i.e. selecting $r = 4$ (while an exact $r = 3$ solution exists) results in errors below 1%. Both methods yield accurate decoupled functions. The implicit method has the advantage of being fully automated and requiring a significantly lower computing time. The computing time of the explicit method scales with the considered range of λ[5]. The accuracy could potentially be increased by using a finer search grid of λ.

Table 1: Relative root-mean-squared error on the function approximation of the toy problem.

		$r = 1$	$r = 2$	$r = 3$	$r = 4$
Implicit	e_1	51.4 %	20.9 %	0.8 %	0.3 %
	e_2	32.1 %	6.2 %	1.0 %	0.4 %
Explicit	e_1	60.7 %	22.8 %	2.8 %	0.2 %
	e_2	32.6 %	4.9 %	1.3 %	0.1 %

Conclusion 1. Both the implicit and the explicit method return accurate decoupled functions. The implicit method has the advantage of being fully automated. It was shown that the use of two filters, a 3-points left finite difference and a 3-points right finite difference, is sufficient for smooth solutions to be obtained.

7.2. Polynomial reduction

In many cases decoupled functions are more efficient parameterisations of the nonlinearity. This is often true for polynomials. In the standard monomial basis, the number of parameters grows combinatorially with the number of inputs m and the degree d. The parameter-count of a decoupled

[5]The scan over λ can be parallelised, reducing computing time.

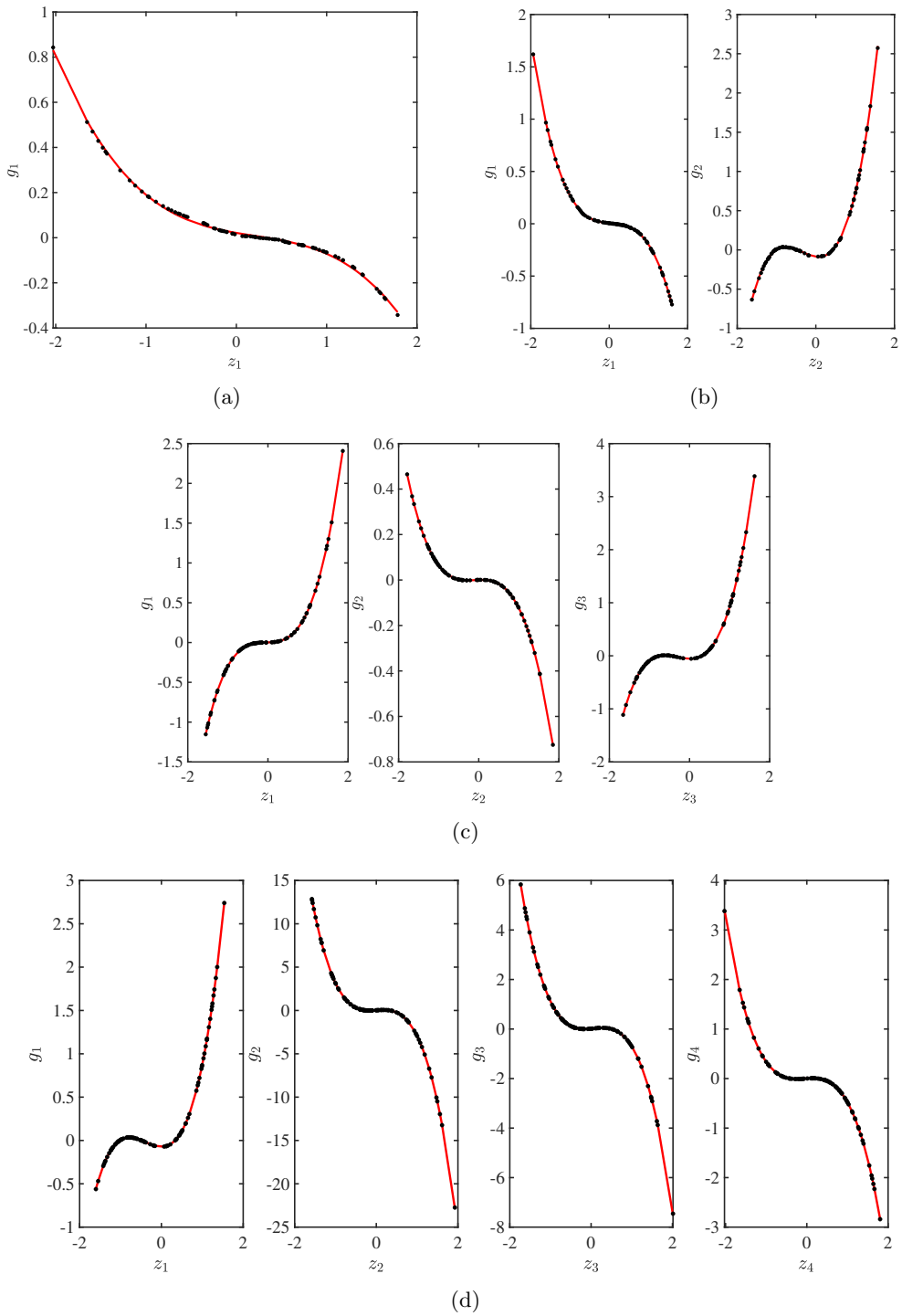
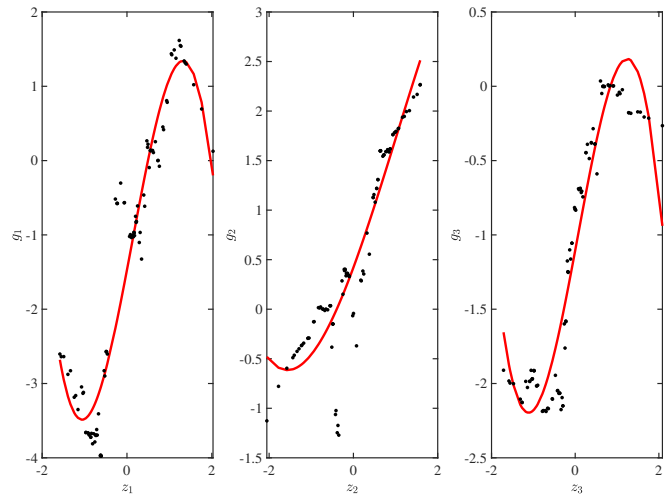
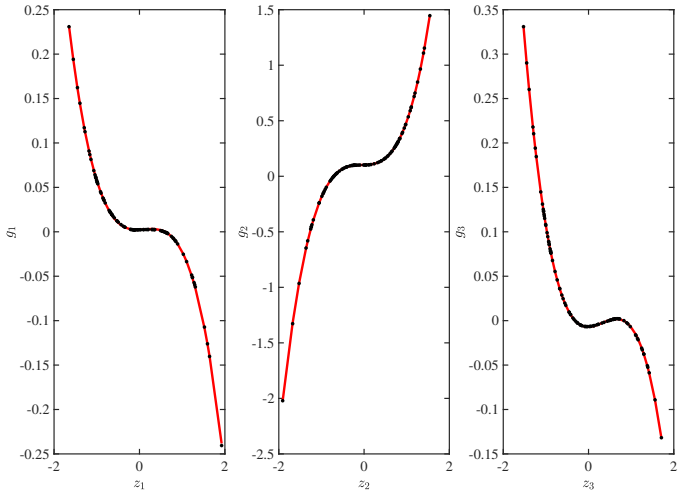


Figure 4: Results on decoupling the toy problem using the implicit FTD method, based on the use of two filters: a 3-points left finite difference and a 3-points right finite difference. (a) $r = 1$ (b) $r = 2$ (c) $r = 3$ (d) $r = 4$. The nonparametric estimates are fitted using polynomials of the third degree. Relative function errors can be consulted in Table 1.



(a)



(b)

Figure 5: Results on decoupling the toy problem using the explicit FTD method. (a) Selecting $\lambda = 0.01$ does not put enough weight on the smoothness objective. (b) A value of $\lambda = 10^{10}$ results in smooth estimates (black markers). In practice λ is found from a line search. Red corresponds to a polynomial fit of the third degree.

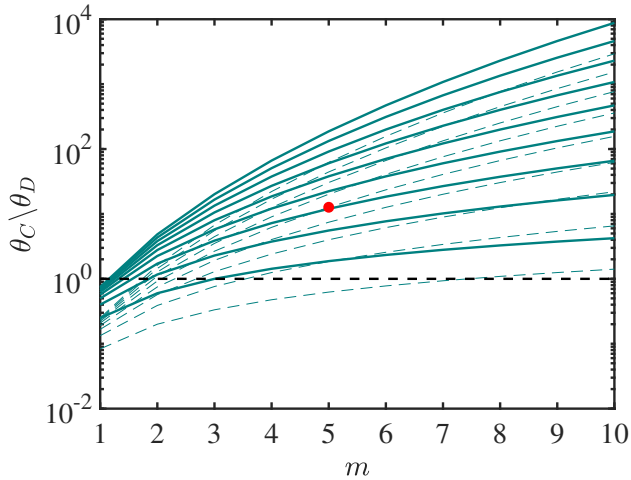


Figure 6: Visualisation of the ratio of the number of parameters required for an expression in the classical monomial basis and the required number for the equivalent decoupled structure. Solid lines corresponds to one-branch functions with a single output, i.e. $n = 1$, for varying number of inputs m and degrees $d = 2, \ldots, 10$ (indicated with different lines, larger d results in larger values of $\theta_C \backslash \theta_D$). The dashed lines depict the scenario for three-branch models. The red dot in corresponds to decoupled functions with $n = 1$, $m = 5$, $d = 5$, and $r = 3$. Without constant terms, such decoupled functions are described by 36 parameters. Expressed in the classical monomial basis, 456 parameters are used, which is more than 12 times as many.

polynomial form, on the other hand, grows only linearly as a function of m and d. The latter is illustrated in Fig. 6. The figure is constructed by considering decoupled polynomial functions with one branch ($r = 1$, these are the solid lines) and one output ($n = 1$) of varying input size m and degrees $d = 2, \ldots, 10$. Each such decoupled polynomial can be translated to the classical basis, resulting in a certain number of parameters. The ratio of the numbers obtained for the coupled and the decoupled forms make up the y-axis. It illustrates that the decoupled representation is more efficient, irrespective of the degree for $m \geq 2$. Considering more branches ($r \geq 1$) shifts the curves downwards, postponing the crossing of the one-to-one ratio indicated by the black dashed line. The scenario for the case of $r = 3$ is indicated by dashed curves.

The red dot in Fig. 6 corresponds to decoupled functions with the following properties: $n = 1$, $m = 5$, $d = 5$, and $r = 3$. Without constant terms, such decoupled functions are described by 36 parameters. Expressed in the classical monomial basis, 456 parameters are used, which is more than 12 times as many. This illustrates that it is often worthwhile to consider a decoupled representation of the nonlinearity.

To put the algorithm to the test, a decoupled polynomial with random parameters (drawn from $\mathcal{N}(0, 1)$) of the described shape is generated. From this function the Jacobian tensor is constructed using 200 operating points, drawn from $\mathcal{U}(-1.5, 1.5)$. Note that given the equivalency of the coupled and the decoupled polynomial form, the Jacobian tensor constructed from either expression is identical. Next, the implicit algorithm is used to retrieve decoupled forms for a scan over r. The default setting using two filters: a 3-points left and a 3-points right filter was selected. The results from a fifth order polynomial parameterisation of the branches are listed in Table 2.

Table 2: Relative root-mean-squared error on the function approximation of the polynomial reduction example. Bottom row corresponds to the ratio of numbers of parameters obtained for the coupled and the decoupled functions.

	$r=1$	$r=2$	$r=3$	$r=4$
Implicit e_1	8.6 %	4.7 %	0.60 %	0.63 %
$\theta_C \backslash \theta_D$	38	19	12.7	9.5

It is important to stress that all coupled polynomial functions can be decoupled (see Section 4). In this example a case was constructed belonging to the subset which require a low number of branches in the decoupled form, leading to significant reductions in the number of parameters. This turns out to be an important class of functions, frequently encountered in practice. The examples in [6, 24, 25, 7] show that nonlinearities found in dynamical models can often be brought back to a low number of underlying characteristic univariate functions. This can lead to insight into the system's behaviour.

Conclusion 2. An important subset of the class of coupled polynomial functions can be represented by a decoupled form containing a low number of univariate polynomial branches. This results in a significant reduction in the number of parameters and may lead to an improved understanding of the relationship.

8. Case study 2: a neural network example

To illustrate the universal approximation properties of the decoupled form, also a non-polynomial example is considered. We will show that popular feed-forward sigmoidal neural networks may be parameterised more efficiently by single-layer decoupled functions with tailored activation (branch) functions.

A feed-forward neural network has a sequential form in which input features cascade through the layers, ultimately forming the output. Denoting the simulated model output by $\boldsymbol{q} \in \mathbb{R}^{n_L}$ and the input at a single time step by $\boldsymbol{p} \in \mathbb{R}^{n_0}$, we have that

$$\boldsymbol{x}^{(0)} = \boldsymbol{p}, \tag{43a}$$

$$\boldsymbol{x}^{(l)} = \boldsymbol{f}^{(l)}(\boldsymbol{x}^{(l-1)}), \quad l = 1, \ldots, L-1 \tag{43b}$$

$$\boldsymbol{q} = \boldsymbol{f}^{(L)}(\boldsymbol{x}^{(L-1)}), \tag{43c}$$

where L denotes the final layer. The applied function in the lth layer usually has the following form

$$\boldsymbol{f}^{(l)}(\boldsymbol{x}^{(l-1)}) = \boldsymbol{\sigma}^{(l)}(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}). \tag{44}$$

If we denote the number of neurons in the l-th layer by n_l, such that $\boldsymbol{x}^{(l)} \in \mathbb{R}^{n_l}$, we have that $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$. The model parameters $\{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}_{l=1}^{L}$ are referred to as the weights $\boldsymbol{W}^{(l)}$ and biases $\boldsymbol{b}^{(l)}$. The nonlinear activation is a vector function $\boldsymbol{\sigma}^{(l)} : \mathbb{R}^{n_l} \mapsto \mathbb{R}^{n_l}$. In classical neural networks, though, all neurons of a single layer are subjected to the same activation.

In this example we will consider sigmoidal activation, such that σ_i is a sigmoidal function. Additionally, $\boldsymbol{f}^{(L)} := \mathbf{W}^{(L)}\boldsymbol{x}^{(L-1)} + \boldsymbol{b}^{(L)}$, also known as linear activation, which is the appropriate choice

for regression problems [26]. An arbitrary neural network is generated by randomly selecting model parameters, i.e. $\{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}_{l=1}^{L} \sim \mathcal{N}(0,1)$. The generated network has the following dimensions: $n_0 = 2$, $n_1 = 15$, $n_2 = 10$, $n_3 = 10$, $n_4 = 5$, $n_5 = 2$, resulting in a total number of 382 parameters. Next we apply the fully automated implicit algorithm (Section 5.2) to retrieve decoupled functions for a range in r. Given that the network is random, it may also contain constant terms on the outputs. To account for the constant terms the procedure of Section 6 is applied.

It was found that the use of additional filters, e.g. a 3-points left, 3-points right, and a 3-points central filter, can be required to facilitate the convergence of the implicit algorithm. This is especially the case for large decoupled functions for which $r > \text{rank}\,(\mathcal{J})$[6], which gives rise to an underdetermined tensor factorisations. The results are presented in Table 3. The mean function approximation error (following from (42)), computed on the selected operating points, is additionally visualised in Fig. 7, illustrating that decoupling can be used to balance model complexity to accuracy. Accurate decoupled functions could be retrieved with errors in the order of 1 % while requiring less than a third of the number of parameters used in the neural network description. A polynomial fit of degree 7 was used in this case.

Table 3: Relative root-mean-squared error on the function approximation of the sigmoidal neural network example. Bottom row corresponds to the ratio of numbers of parameters obtained for the network and the decoupled functions.

	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$	$r=7$	$r=8$	$r=9$	$r=10$
e_1	34.2 %	9.2 %	4.7 %	1.6 %	1.0 %	0.8 %	3.7 %	2.8 %	1.1 %	1.4 %
e_2	92.8 %	88.0 %	12.1 %	5.6 %	5.3 %	3.8 %	5.0 %	3.0 %	2.5 %	2.1 %
mean e	63.5 %	48.6 %	8.4 %	3.6 %	3.2 %	2.3 %	4.3 %	2.9 %	1.8 %	1.7 %
$\theta_{NN} \backslash \theta_D$	27.3	15.3	10.6	8.1	6.6	5.5	4.8	4.2	3.7	3.4

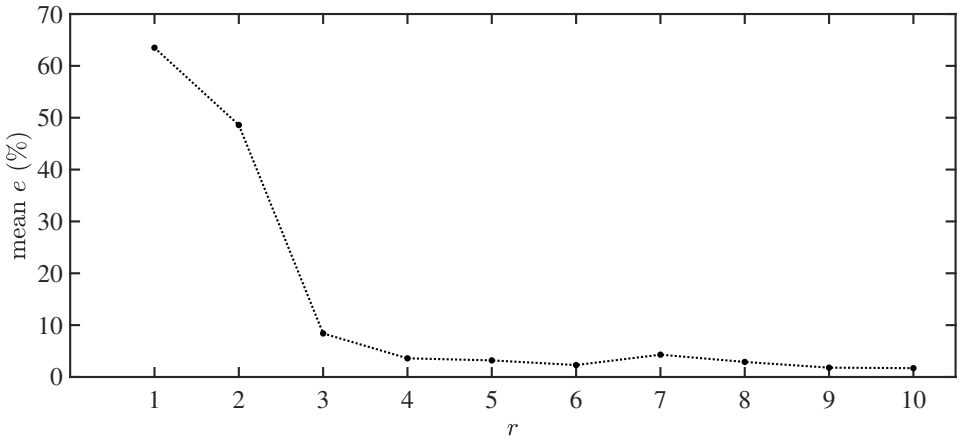
Conclusion 3. Decoupling is a generic tool. The universal approximation theorem states that all continuous real functions may be approximated arbitrary well by a decoupled form. The required complexity of the decoupled function (number of branches) depends on the underlying relationship alone, and not on the family of functions used to express it.

Conclusion 4. Large decoupling problems, for which $r > \text{rank}\,(\mathcal{J})$, give rise to underdetermined tensor factorisation. In such cases additional filters may be required to restrict the solutions of the implicit algorithm to a meaningful subset.

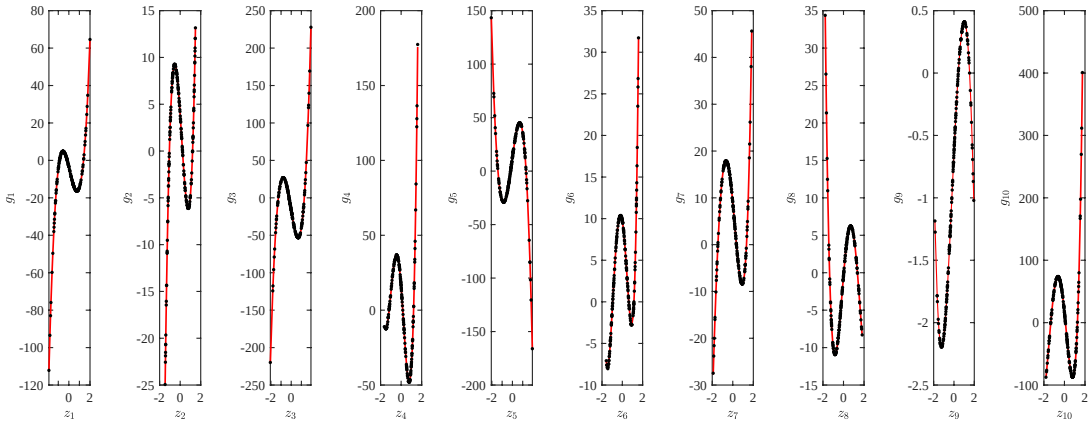
9. Case study 3: NARX models

An important benefit from function decoupling is that also nonlinear dynamical models, as they appear in system identification, can be handled. Note that nonlinear (data-driven) dynamical models rely on multivariate mapping functions as description of the nonlinearity. This is the case for nonlinear state-space models, where either a multivariate polynomial [27, 28], or a neural network is used [29], but also for NARX models [23].

[6]The rank of a tensor is here defined as the minimal number of rank-one terms required to minimise (9).



(a)



(b)

Figure 7: (a) Mean function approximation error when decoupling the sigmoidal neural network example. (b) Results for a decoupling using $r = 10$ branches. The functions are fitted using a 7th order polynomial.

9.1. Single-output Polynomial NARX models

In this section we will show that the tools developed in this work allow to decouple NARX models in general, and single-output NARX models in particular. Single-output functions are out of the scope of the original decoupling method of [8]. Given that a collection of Jacobians of a single-output function results in a matrix, rather than a third-order tensor, one cannot rely on the uniqueness properties of tensor decompositions, which is the cornerstone of the method of [8]. An alternative approach for decoupling single-output NARX models was proposed in [12]. The procedure of [12] resorts to a decomposition of the Hessian, combined with polynomial constraints on one of the factors. The benefits of the tools presented in this work lie in the fact that, both the expensive computation of the Hessian, as well as the polynomial constraints can be avoided. Recall that in this work non-parametric estimates of g_i are obtained, i.e. without having to predefine a function family.

NARX models describe dynamical nonlinear behaviour by relating the current output sample to both past output samples, and current and past input samples. Let

$$\boldsymbol{p}(t) \coloneqq \{u(t), u(t-1), \ldots, u(t-n_u), y(t-1), \ldots, y(t-n_y)\}. \tag{45}$$

A general single-output NARX model then assumes the form [23]

$$y(t) = F(\boldsymbol{p}(t)) + e(t), \tag{46}$$

with u the input, and y the measured output. $F : \mathbb{R}^{n_u+n_y+1} \to \mathbb{R}$ is a static multiple-input single-output (MISO) nonlinear function and $e(t)$ is an equation error which is assumed to be a sequence of independent identically distributed (IID) random variables. The function F may be described by any function family, e.g. a neural network. Often, however, a basis expansion is preferred. In that case, a direct estimate of the model parameters follows from a linear regression problem by using the measured outputs in the regressor, as is done in (45) (minimising the equation error). A popular choice is the polynomial basis, leading to so-called Polynomial-NARX or P-NARX models.

The downside of P-NARX models is that they suffer from the curse of dimensionality. As was discussed in section 7.2, the number of model parameters grows combinatorially with the number of delayed inputs, outputs, and nonlinear degree, contrary to the linear growth with r and the degree in case of a decoupled polynomial.

9.2. Benchmark problem: the forced Duffing oscillator

The decoupling of single-output P-NARX models is demonstrated on a nonlinear benchmark data set of the forced Duffing oscillator. The idealised system equation reads

$$m\ddot{y}(t) + c\dot{y}(t) + k(y(t))y(t) = u(t), \tag{47}$$

with mass m, viscous damping c, and a nonlinear spring $k(y(t))$. The presumed displacement, $y(t)$, is considered the output and the presumed force, $u(t)$, is considered the input. Overdots denote the derivative with respect to time. The system can be interpreted as a cubic hardening spring given the position-dependent stiffness

$$k(y(t)) = \alpha + \beta y^2(t). \tag{48}$$

Data are obtained from an analogue electrical circuitry. **The training set** consists of 9 realisations of a random-phase odd multisine [30]. The period of the multisine is $1/f_0$ with $f_0 = f_s/8192$ Hz and $f_s \approx 610$ Hz. The number of excited harmonics is $L = 1342$ resulting in an $f_{\max} \approx 200$

Hz. Each multisine realisation is given a unique set of phases that are independent and uniformly distributed in $[0, 2\pi[$. The signal to noise ratio at the output is estimated at approximately 40 dB. As **test set**, a filtered Gaussian noise sequence of the same band width and with a linearly increasing amplitude is used.

The data are part of three benchmark data sets for nonlinear system identification described in [31] and used in a number of works among which [12, 32, 33, 34, 35].

9.2.1. Reference P-NARX model

Using the System Identification toolbox in MATLAB a P-NARX model is estimated with the following properties: $n_u = 1$, $n_y = 3$, and $d = 3$. All cross-term monomials where included leading to a model with 55 parameters. The model is estimated in 'prediction'-mode, i.e. by minimising the equation error in (46). The accuracy of the model will, however, be evaluated on the basis of a simulation error, i.e. by reinserting the modelled output back into the regressor

$$\boldsymbol{p}_s(t) \coloneqq \{u(t), u(t-1), \ldots, u(t-n_u), \hat{y}_s(t-1), \ldots, \hat{y}_s(t-n_y)\}, \tag{49}$$

$$\hat{y}_s(t) = F(\boldsymbol{p}_s(t)), \tag{50}$$

where the subscript s refers to the simulation setting. A relative root-mean-square simulation error may then be defined as

$$e_{\text{rms}} = \frac{\sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\hat{y}_s(t) - y(t)\right)^2}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(y(t) - \mathbb{E}(y(t))\right)^2}} \times 100. \tag{51}$$

The estimation process returns an accurate P-NARX model yielding a simulation error e_{rms} of 1.01% on the test set.

9.2.2. Decoupled P-NARX model

In this case study the use of the explicit smoothness method (section 5.3) is demonstrated. We will construct decoupled NARX models of the form

$$F(\boldsymbol{p}_s(t)) \approx \boldsymbol{W}\boldsymbol{g}\left(\boldsymbol{V}^\top \boldsymbol{p}_s(t)\right). \tag{52}$$

Given that the presented method allows to treat r as a design parameter, a range of models of varying complexity can be constructed. We will consider the range $r = 1, \ldots, 6$. The explicit method additionally requires a scan over the hyperparameter λ. A coarse search over $\sqrt{\lambda} = 10^{-1}, 10^0, 10^1, 10^2, 10^3, 10^5$, is conducted. The univariate branches are in this case parameterised using third-degree polynomial functions. For every value of r, the decoupled model yielding the best performance on the training data (according to (51)) is selected. The models are compared to the reference P-NARX model on the basis of the test set performance in Fig. 8. Observe that decoupled models with $r \geq 3$, denoted by 'o' markers, result in a performance similar to that of the reference model (indicated by the red line).

If needed, the decoupled model can be subjected to a post-optimisation step. In that case the model parameters are additionally tuned on the basis of the training data given a mean-squared error cost function

$$\underset{\boldsymbol{W},\boldsymbol{V},\boldsymbol{\theta}}{\arg\min} \sum_{t=1}^{N_T} \left(y(t) - \boldsymbol{W}\boldsymbol{g}\left(\boldsymbol{V}^\top \boldsymbol{p}_s(t)\right)\right)^2, \tag{53}$$

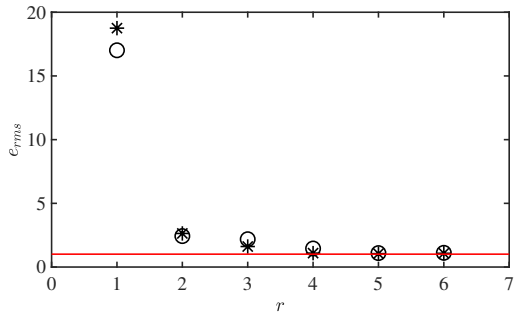


Figure 8: Test set performance of the decoupled NARX models, obtained using the explicit decoupling algorithm and starting from the reference P-NARX model of section 9.2.1. 'o' indicate the performance of the models without post-optimistion, '*' denotes the post-optimised models, found according to (53). The red line corresponds to the performance of the reference P-NARX model and corresponds to a value of $e_{\text{rms}} = 1.01\%$.

with $\boldsymbol{\theta}$ storing the polynomial coefficients of $\boldsymbol{g}$, and N_T the length of the training set. The test set performance of the post-optimised models is indicated by '*' in Fig.8. Notice that additional tuning on the training set is not guaranteed to improve the test set result.

The benefit of obtaining a decoupled NARX model is twofold:

1. The decoupled model with $r = 3$ requires only 30 parameters (compared to 55 for the reference model), while resulting in a similar performance. The complexity of the model can, moreover, easily be balanced to the accuracy. For a given application the accuracy of the $r = 2$ model might still be sufficient, resulting in an even larger model reduction.

2. Given that the nonlinear elements are now described by univariate nonlinearities, they may easily be visualised, as is done in Fig. 9. The functions reveal a dominantly cubic relationship in branch 1 and 3, and a dominantly linear form in 2. This relates to the physical model (47), containing both linear and cubic stiffness.

A similar test set performance was obtained for the competing method of [12], yielding a value of 1.78% given the same settings, i.e. $r = 4$, $n_u = 1$ and $n_y = 3$ [24], and a value of 0.9% in case of a slightly larger model with $r = 4$, $n_u = 2$ and $n_y = 3$ [12].

Conclusion 5. Decoupling can be used to process dynamical models. The result is a reduced model with tractable nonlinear elements. A visualisation of the nonlinear functions can provide insight into the underlying physical system.

Conclusion 6. The filtered decoupling procedures, presented in this work, allow to decoupled single-output functions, which was out of the scope of the original method of [8]. As a result, also single-output NARX models can be handled.

10. Discussion

The examples presented in this work illustrate that a decoupled function can be a useful alternative representation of the classical basis expansion or neural network. It is, however, important to

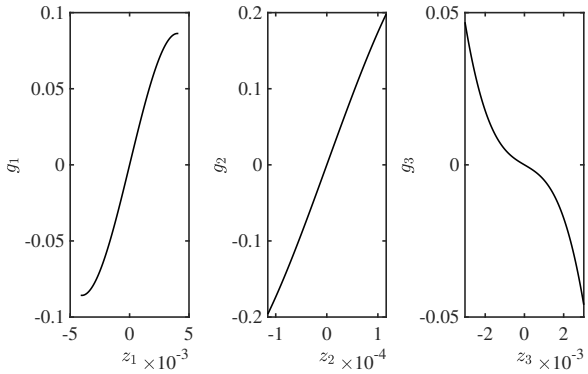


Figure 9: Nonlinear branch functions of the post-optimised $r = 3$ decoupled NARX model. The functions reveal a dominantly cubic relationship in branch 1 and 3, and a dominantly linear form in 2. This relates to the physical model (47), containing both linear and cubic stiffness.

note that decoupling techniques are designed to be a subsequent step in the modelling process. The decoupled form is distilled from a *coupled* function. The sequence of the procedure is imperative for the following reasons:

- Direct estimation of a decoupled function would involve solving a hard nonlinear optimisation problem (output error minimisation). Moreover, a parameterisation of the branches would be required from the start, fixing the function family. Without proper initialisation, the practical use of decoupled forms would be limited to small functions, i.e. with few branches.

- Classical regression tools are powerful because they rely on well established parameter estimation procedures. Basis expansions result in linear problems, allowing solutions to be found in one step. Neural networks exploit the topology of high dimensional cost functions. It can be shown that it becomes exponentially unlikely for the eigenvalues of the Hessian at a critical point to become all positive (or negative). Therefore most critical points are saddle points, rather than local minima [36]. Moreover, if you do get trapped at a saddle point, the analysis seems to show that the large number of saddle points have very similar values of the objective function, explaining why neural networks are not very sensitive to initialisation [37].

It is therefore proposed to exploit the favourable optimisation properties of the classical methods and promote the use of decoupled forms only in a second step. Having obtained a decoupled function, it may be beneficial to consider it as an initialisation and subject it to further tuning using nonlinear optimisation. Such a post-optimisation step, based on input-output data, is useful since it may remove any bias which was present in the originally obtained coupled function.

11. Conclusion

Decoupling multivariate functions is a generic tool which may be applied to all function families. In many cases a more efficient parameterisation of the nonlinear relationship is retrieved. Moreover, the structured decoupled form may reveal a set of characteristic functions which lie at the heart

of the nonlinear mapping, potentially leading to insight. Two algorithms based on filtered tensor decompositions of first order derivative information were introduced. The methods are nonparametric, allowing decoupled functions of any type to emerge. A parameterised form is only obtained in a final isolated step. Decoupling provides a means for model reduction, directly applicable to i.a. nonlinear system identification and machine learning.

References

[1] A. Kratsios, The universal approximation property: characterization, construction, representation, and existence, Annals of Mathematics and Artificial Intelligence.

[2] G. Cybenko, Approximation by superpositions of a Sigmoidal function, Math. Control Signals Systems 2 (1989) 303–314.

[3] T. Hastie, J. Friedman, R. Tbshirani, Basis expansion and regularization. In: The elements of statistical learning, Springer series in statistics, Springer, New York, 2001.

[4] Z. Lu, H. Pu, F. Wang, Z. Hu, L. Wang, The expressive power of neural networks: A view from the width, in: 31st Conference on Neural Information Processing Systems (NIPS 2017), 2017.

[5] S. Brunton, J. L. Proctor, N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proceedings of the National Academy of Sciences of the United States of America (PNAS 113 (15) (2016) 3932–3937.

[6] J. Decuyper, K. Tiels, M. C. Runacres, J. Schoukens, Retrieving highly structured models starting from black-box nonlinear state-space models using polynomial decoupling, Mechanical Systems And Signal Processing 146.

[7] P. Z. Csurcsia, J. Decuyper, B. Renczes, T. De Troyer, Nonlinear Modelling of an F16 Benchmark Measurement, in: Conference Proceedings of the Society for Experimental Mechanics Series, 2022.

[8] P. Dreesen, M. Ishteva, J. Schoukens, Decoupling multivariate polynomials using first-order information, SIAM Journal on Matrix Analysis and Applications 36 (2) (2015) 864–879.

[9] T. G. Kolda, B. W. Bader, Tensor decomposition and applications, SIAM Rev. 51 (3) (2009) 455–500.

[10] K. Usevich, P. Dreesen, M. Ishteva, Decoupling multivariate polynomials: Interconnections between tensorizations, Journal of Computational and Applied Mathematics 363 (1) (2020) 22–34.

[11] J. B. Kruskal, Rank decomposition, and uniqueness for 3-way and N-way arrays, Elsevier Science Publishers B.V., 1989.

[12] K. Karami, D. Westwick, J. Schoukens, Applying polynomial decoupling methods to the polynomial NARX model, Mechanical Systems And Signal Processing 148 (2021) 107134.

[13] P. Dreesen, J. De Geeter, M. Ishteva, Decoupling multivariate functions using second-order information and tensors, in: Proc. 14th International Conference on Latent Variable Analysis and Signal Separation LVA/ICA 2018, Vol. 10891, Springer, 2018, pp. 79–88.

[14] W. Rudin, Principles of mathematical analysis, McGraw-Hill, 1976.

[15] P. Diaconis, M. Shahshahani, On nonlinear functions of linear combinations, SIAM Journal on Scientific and Statistical Computing 5 (1) (1984) 175–191.

[16] J. Decuyper, P. Dreesen, J. Schoukens, M. Runacres, K. Tiels, Decoupling multivariate polynomials for nonlinear state-space models, IEEE Control Systems Letters 3 (3) (2019) 745–750.

[17] J. Decuyper, K. Tiels, S. Weiland, J. Schoukens, Decoupling multivariate functions using a non-parametric Filtered-CPD approach, IFAC-PapersOnLine 54 (7) (2021) 451–456.

[18] J. D. Carroll, J. J. Chang, Analysis of individual differences in multidimensional scaling via an N-way generalization of Eckart-Young decomposition, Psychometrika 35 (283-319).

[19] R. A. Harshman, Foundations of the PARAFAC procedure: Models and conditions for an explanatory multi-modal factor analysis, UCLA Working Papers in Phonetics 16 (1970) 1–84.

[20] K. Levenberg, A method for the solution of certain non-linear problems in least squares, Quarterly of Applied Mathematics 2 (2) (1944) 164–168.

[21] C. Navasca, L. De Lathauwer, S. Kindermann, Swamp reducing technique for tensor decomposition, in: 16th European Signal Processing Conference (EUSIPCO 2008), 2008.

[22] B. Renczes, J. Decuyper, M. C. Runacres, A fast analytical solution to the filtered canonical polyadic decomposition problem, in: IEEE International Instrumentation and Measurement Technology Conference I2MTC, 2022.

[23] S. Billings, Nonlinear System Identification: NARMAX Methods in the Time, Frequency and Spatio-Temporal Domains, Wiley, 2013.

[24] J. Decuyper, D. Westwick, K. Karami, J. Schoukens, Decoupling P-NARX models using filtered CPD, IFAC-PapersOnLine 54 (7) (2021) 661–666.

[25] P. Csurcsia, T. De Troyer, An empirical study on decoupling pnlss models illustrated on an airplane, IFAC-PapersOnLine 54 (7) (2021) 673–678.

[26] I. Goodfellow, Y. Bengio, A. Courville, Deeep learning, The MIT Press, 2016.

[27] J. Decuyper, K. Tiels, J. Schoukens, PNLSS Toolbox 1.0, IFAC-PapersOnLine 54 (7) (2021) 374–378.

[28] J. Paduart, L. Lauwers, J. Swevers, K. Smolders, J. Schoukens, R. Pintelon, Identification of nonlinear systems using polynomial nonlinear state space models, Automatica 46 (2010) 647–657.

[29] M. Schoukens, Improved initialization of state-space artificial neural networks, in: European Control Conference (ECC), 2021, p. https://doi.org/10.48550/arXiv.2103.14516.

[30] R. Pintelon, Detecting and quantifying the nonlinear and time-variant effects in frf measurements using periodic excitations, IEEE Transactions on Instrumentation and Measurement 62 (12) (2013) 3361–3373.

[31] T. Wigren, J. Schoukens, Three free data sets for development and benchmarking in nonlinear system identification, in: European Control Conference (ECC), Zurich, Switzerland, 2013, pp. 2933–2938.

[32] J.-P. Noël, J. Schoukens, Grey-box state-space identification of nonlinear mechanical vibrations, International Journal of Control 91 (5) (2018) 1118–1139.

[33] L. Ljung, Q. Zhang, P. Lindskog, A. Juditski, Modeling a non- linear electric circuit with black box and grey box models, in: Proc. IFAC Symposium on Nonlinear Control Systems (NOLCOS2004), 2004, pp. 543–548.

[34] L. Sragner, J. Schoukens, G. Horvath, Modeling of slightly nonlinear systems: a neural network approach, in: Proc. IFAC Symposium on Nonlinear Control Systems (NOLCOS2004), 2004, pp. 531–536.

[35] J. Schoukens, L. Ljung, Nonlinear system identification: a user-oriented road map, IEEE Control Systems Magazine 39 (6) (2019) 28–99.

[36] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, in: Proceedings of the 27th International Conference on Neural Information Processing Systems, Vol. 2, 2014, pp. 2933–2941.

[37] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.

[38] A. K. Singh, B. S. Bhadauri, Finite difference formulae for unequal subintervals using Lagrange's interpolation formula, Int. Journal of Math. Analysis 3 (17) (2009) 815–827.

[39] J. L. Lagrange, Leçons élémentaires sur les mathématiques, Journal de l'École polytechnique, VIIe et VIIIe cahiers 7 (183-288).

[40] E. Suli, D. F. Mayers, An introduction to numerical analysis, Cambridge University Press, 2003.

Appendix A. Non-equidistant finite differences

A finite difference expression for non-equidistantly spaced points can be obtained through Lagrange interpolation [38]. Let $\boldsymbol{z}_s \in \mathbb{R}^N$ be a vector of z-values, sorted in ascending order and let $\boldsymbol{g}_s \in \mathbb{R}^N$ be the vector of corresponding evaluations of the function $g(z)$, sorted according to z. Consider $\boldsymbol{z}_w$ to be a window of k neighbouring elements such that $\boldsymbol{z}_w \subset \boldsymbol{z}_s$ and let $\boldsymbol{g}_w$ be the corresponding window of function evaluations such that $\boldsymbol{g}_w \subset \boldsymbol{g}_s$. The Lagrange polynomial is the polynomial of lowest degree which interpolates all k points within the window [39, 40]. It is a linear combination of basis functions

$$L(z) \coloneqq \sum_{j=1}^{k} \boldsymbol{g}_w[j] l_j(z), \tag{A.1}$$

where l_j are known as Lagrange basis polynomials

$$l_j(z) \coloneqq \prod_{\substack{1 \leq i \leq k \\ i \neq j}} \frac{z - \boldsymbol{z}_w[i]}{\boldsymbol{z}_w[j] - \boldsymbol{z}_w[i]}. \tag{A.2}$$

At the grid points $\boldsymbol{z}_w$, the basis polynomials have the following two properties:

1. For all $s \neq j$ the product will be zero since the numerator $\boldsymbol{z}_w[s] - \boldsymbol{z}_w[i]$ attains zero at $s = i$.

$$\forall (s \neq j) : l_j(\boldsymbol{z}_w[s]) = \prod_{\substack{1 \leq i \leq k \\ i \neq j}} \frac{\boldsymbol{z}_w[s] - \boldsymbol{z}_w[i]}{\boldsymbol{z}_w[j] - \boldsymbol{z}_w[i]} = 0. \tag{A.3}$$

2. On the other hand

$$l_j(\boldsymbol{z}_w[j]) = \prod_{\substack{1 \leq i \leq k \\ i \neq j}} \frac{\boldsymbol{z}_w[j] - \boldsymbol{z}_w[i]}{\boldsymbol{z}_w[j] - \boldsymbol{z}_w[i]} = 1. \tag{A.4}$$

Given that all basis polynomials are zero when $s \neq j$, and since they equal to one for $s = j$, it follows that $L(\boldsymbol{z}_w[j]) = \boldsymbol{g}_w[j]$, meaning that $L(z)$ passes exactly through all k points in the window.

Having found a function which relates neighbouring points, also a derivative relationship between neighbouring points (i.e. a finite difference) may be obtained. A first order derivative expression is given by

$$L^{(1)}(z) \coloneqq \sum_{j=1}^{k} \boldsymbol{g}_w[j] l_j^{(1)}(z), \tag{A.5}$$

which is a weighted sum with the weights following from $l_j^{(1)}(z) \coloneqq \frac{dl_j(z)}{dz}$

$$l_j^{(1)}(z) \coloneqq \sum_{\substack{s=1 \\ s \neq j}}^{k} \left[\frac{1}{\boldsymbol{z}_w[j] - \boldsymbol{z}_w[s]} \prod_{\substack{i=1 \\ i \neq (s,j)}}^{k} \frac{z - \boldsymbol{z}_w[i]}{\boldsymbol{z}_w[j] - \boldsymbol{z}_w[i]} \right]. \tag{A.6}$$

The first order derivative based on Lagrange polynomials, i.e. $L^{(1)}(z)$, is illustrated on a sampled sine function in Fig. A.10.

Notice that the linear combination, expressed by (A.5), corresponds to the operations encoded in the rows of the matrix $\boldsymbol{D}$ ((14)). When computing a 3-points finite difference, $k = 3$, the weights $l_j^{(1)}(z)$ with $j = 1, \ldots, k$ form the elements of $\boldsymbol{D}$. Contrary to (14), all weights may differ from one another since they are computed on the basis of the non-equidistant spacing of the k points in $\boldsymbol{z}_w$. Going from one row to the next may be seen as sliding the window of the finite difference operator over the z-axis. The positioning of the window allows to switch from forward to central to backward differencing, additionally also the length may be changed. This flexibility enables to compute a number of alternative finite difference filters, which are all approximations of the first order derivative. Consider the following examples:

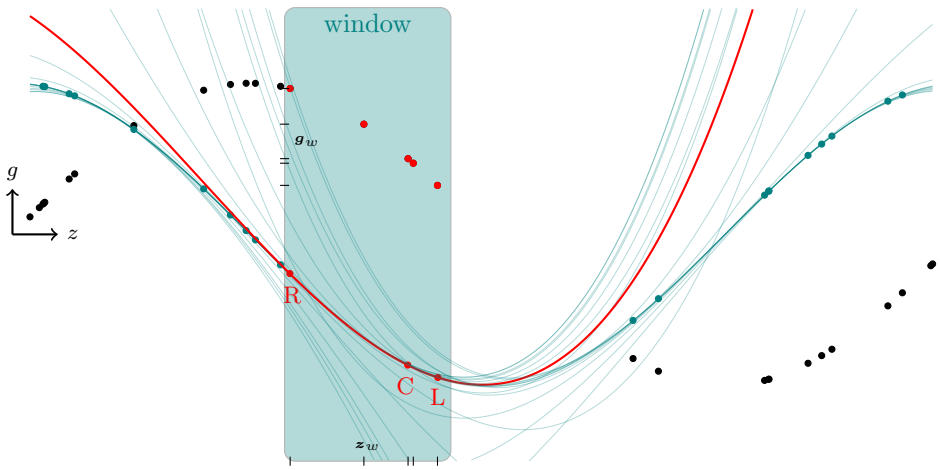


Figure A.10: Illustration of the use of Lagrange polynomials to compute non-equidistant finite differences. Black markers indicate evaluations of a sine function on a non-equidistant grid of points. Based on a sliding window of 5 points the Lagrange polynomial is constructed. The derivative of this polynomial, i.e. $L^{(1)}(z)$ is indicated by a thin solid line (for every window). Evaluating $L^{(1)}(z)$ at the grid points results in a finite difference approximation. $L^{(1)}(z)$ corresponding to the indicated window is highlighted in red. If this curve is used to estimate the derivative in the most forward point of the window a left (or forward) finite difference is obtained. Analogously a central and a right (or backward) finite difference may be obtained. Collecting, for example, the central difference estimate of every window reveals the cosine function indicated by teal markers.

- **3-points central finite difference filter**

The ith row of a 3-points ($k = 3$) central differencing filter of size N, $\boldsymbol{D}_C$, is based on the window $\boldsymbol{z}_{w_i} = [\boldsymbol{z}_s[i-1] \quad \boldsymbol{z}_s[i] \quad \boldsymbol{z}_s[i+1]]^\top$. The weights are accordingly found by evaluating the basis functions $l_j^{(1)}(z)$ in the central point of the window, i.e. $l_j^{(1)}(\boldsymbol{z}_{w_i}[2])$ with $j = 1, \ldots, k$. The following filter is obtained:

$$\boldsymbol{D}_C \coloneqq \begin{bmatrix} l_1^{(1)}(\boldsymbol{z}_{w_1}[1]) & l_2^{(1)}(\boldsymbol{z}_{w_1}[1]) & l_3^{(1)}(\boldsymbol{z}_{w_1}[1]) & 0 & \ldots & 0 \\ l_1^{(1)}(\boldsymbol{z}_{w_2}[2]) & l_2^{(1)}(\boldsymbol{z}_{w_2}[2]) & l_3^{(1)}(\boldsymbol{z}_{w_2}[2]) & 0 & \ldots & 0 \\ 0 & l_1^{(1)}(\boldsymbol{z}_{w_3}[2]) & l_2^{(1)}(\boldsymbol{z}_{w_3}[2]) & l_3^{(1)}(\boldsymbol{z}_{w_3}[2]) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & l_1^{(1)}(\boldsymbol{z}_{w_{N-1}}[2]) & l_2^{(1)}(\boldsymbol{z}_{w_{N-1}}[2]) & l_3^{(1)}(\boldsymbol{z}_{w_{N-1}}[2]) \\ 0 & \ldots & 0 & l_1^{(1)}(\boldsymbol{z}_{w_N}[3]) & l_2^{(1)}(\boldsymbol{z}_{w_N}[3]) & l_3^{(1)}(\boldsymbol{z}_{w_N}[3]) \end{bmatrix}. \tag{A.7}$$

The first and the last row face a boundary and are treated as 3-points right (backward) and 3-points left (forward) filters, correspondingly. For convenience, an explicit expression is provided for the 3 non-zero entries of th ith row

$$l_1^{(1)}\left(\boldsymbol{z}_{w_i}[2]\right) = \frac{\boldsymbol{z}_{w_i}[2] - \boldsymbol{z}_{w_i}[3]}{\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[2]\right)\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[3]\right)}, \tag{A.8}$$

$$l_2^{(1)}\left(\boldsymbol{z}_{w_i}[2]\right) = \frac{\boldsymbol{z}_{w_i}[1] - 2\boldsymbol{z}_{w_i}[2] + \boldsymbol{z}_{w_i}[3]}{\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[2]\right)\left(\boldsymbol{z}_{w_i}[2] - \boldsymbol{z}_{w_i}[3]\right)}, \tag{A.9}$$

$$l_3^{(1)}\left(\boldsymbol{z}_{w_i}[2]\right) = \frac{\boldsymbol{z}_{w_i}[2] - \boldsymbol{z}_{w_i}[1]}{\left(\boldsymbol{z}_{w_i}[3] - \boldsymbol{z}_{w_i}[2]\right)\left(\boldsymbol{z}_{w_i}[3] - \boldsymbol{z}_{w_i}[1]\right)}. \tag{A.10}$$

- **3-points left finite difference filter**

The ith row of a 3-points ($k = 3$) left differencing filter of size N, $\boldsymbol{D}_L$, is based on the window $\boldsymbol{z}_{w_i} = [\boldsymbol{z}_s[i-2] \quad \boldsymbol{z}_s[i-1] \quad \boldsymbol{z}_s[i]]^\top$. The weights are accordingly found by evaluating the basis functions $l_j^{(1)}(z)$ in the most forward point of the window, i.e. $l_j^{(1)}(\boldsymbol{z}_{w_i}[3])$ with

$j = 1, \ldots, k$. The following filter is obtained:

$$\boldsymbol{D}_L := \begin{bmatrix} l_1^{(1)}(\boldsymbol{z}_{w_1}[1]) & l_2^{(1)}(\boldsymbol{z}_{w_1}[1]) & l_3^{(1)}(\boldsymbol{z}_{w_1}[1]) & 0 & \ldots & 0 \\ l_1^{(1)}(\boldsymbol{z}_{w_2}[2]) & l_2^{(1)}(\boldsymbol{z}_{w_2}[2]) & l_3^{(1)}(\boldsymbol{z}_{w_2}[2]) & 0 & \ldots & 0 \\ l_1^{(1)}(\boldsymbol{z}_{w_3}[3]) & l_2^{(1)}(\boldsymbol{z}_{w_3}[3]) & l_3^{(1)}(\boldsymbol{z}_{w_3}[3]) & 0 & \ldots & 0 \\ 0 & l_1^{(1)}(\boldsymbol{z}_{w_4}[3]) & l_2^{(1)}(\boldsymbol{z}_{w_4}[3]) & l_3^{(1)}(\boldsymbol{z}_{w_4}[3]) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & l_1^{(1)}(\boldsymbol{z}_{w_N}[3]) & l_2^{(1)}(\boldsymbol{z}_{w_N}[3]) & l_3^{(1)}(\boldsymbol{z}_{w_N}[3]) \end{bmatrix}. \quad \text{(A.11)}$$

Given the boundary, the first row is treated as a backward scheme, the second as a central scheme and starting from the third a forward (or left finite difference) is obtained. Also here, an explicit expression is provided for the 3 non-zero entries of th ith row

$$l_1^{(1)}\left(\boldsymbol{z}_{w_i}[3]\right) = \frac{\boldsymbol{z}_{w_i}[3] - \boldsymbol{z}_{w_i}[2]}{\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[3]\right)\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[2]\right)}, \quad \text{(A.12)}$$

$$l_2^{(1)}\left(\boldsymbol{z}_{w_i}[3]\right) = \frac{\boldsymbol{z}_{w_i}[3] - \boldsymbol{z}_{w_i}[1]}{\left(\boldsymbol{z}_{w_i}[2] - \boldsymbol{z}_{w_i}[3]\right)\left(\boldsymbol{z}_{w_i}[2] - \boldsymbol{z}_{w_i}[1]\right)}, \quad \text{(A.13)}$$

$$l_3^{(1)}\left(\boldsymbol{z}_{w_i}[3]\right) = \frac{\boldsymbol{z}_{w_i}[1] - 2\boldsymbol{z}_{w_i}[3] + \boldsymbol{z}_{w_i}[2]}{\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[3]\right)\left(\boldsymbol{z}_{w_i}[3] - \boldsymbol{z}_{w_i}[2]\right)}. \quad \text{(A.14)}$$

- **3-points right finite difference filter**

  The ith row of a 3-points ($k = 3$) right differencing filter of size N, $\boldsymbol{D}_R$, is based on the window $\boldsymbol{z}_{w_i} = [\boldsymbol{z}_s[i] \quad \boldsymbol{z}_s[i+1] \quad \boldsymbol{z}_s[i+2]]^\top$. The weights are accordingly found by evaluating the basis functions $l_j^{(1)}(z)$ in the backward point of the window, i.e. $l_j^{(1)}(\boldsymbol{z}_{w_i}[1])$ with $j = 1, \ldots, k$.

The following filter is obtained:

$$\boldsymbol{D}_R \coloneqq \begin{bmatrix} l_1^{(1)}(\boldsymbol{z}_{w_1}[1]) & l_2^{(1)}(\boldsymbol{z}_{w_1}[1]) & l_3^{(1)}(\boldsymbol{z}_{w_1}[1]) & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & l_1^{(1)}(\boldsymbol{z}_{w_{N-3}}[1]) & l_2^{(1)}(\boldsymbol{z}_{w_{N-3}}[1]) & l_3^{(1)}(\boldsymbol{z}_{w_{N-3}}[1]) & 0 \\ 0 & \dots & 0 & l_1^{(1)}(\boldsymbol{z}_{w_{N-2}}[1]) & l_2^{(1)}(\boldsymbol{z}_{w_{N-2}}[1]) & l_3^{(1)}(\boldsymbol{z}_{w_{N-2}}[1]) \\ 0 & \dots & 0 & l_1^{(1)}(\boldsymbol{z}_{w_{N-1}}[2]) & l_2^{(1)}(\boldsymbol{z}_{w_{N-1}}[2]) & l_3^{(1)}(\boldsymbol{z}_{w_{N-1}}[2]) \\ 0 & \dots & 0 & l_1^{(1)}(\boldsymbol{z}_{w_N}[3]) & l_2^{(1)}(\boldsymbol{z}_{w_N}[3]) & l_3^{(1)}(\boldsymbol{z}_{w_N}[3]) \end{bmatrix}. \tag{A.15}$$

Given the boundary, the last row is treated as a forward scheme, the second to last as a central scheme and starting from the third to last a backward (or right finite difference) is obtained. An explicit expression is provided for the 3 non-zero entries of th ith row

$$l_1^{(1)}\left(\boldsymbol{z}_{w_i}[1]\right) = \frac{-\boldsymbol{z}_{w_i}[2] + 2\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[3]}{\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[2]\right)\left(\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[3]\right)}, \tag{A.16}$$

$$l_2^{(1)}\left(\boldsymbol{z}_{w_i}[1]\right) = \frac{\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[3]}{\left(\boldsymbol{z}_{w_i}[2] - \boldsymbol{z}_{w_i}[1]\right)\left(\boldsymbol{z}_{w_i}[2] - \boldsymbol{z}_{w_i}[3]\right)}, \tag{A.17}$$

$$l_3^{(1)}\left(\boldsymbol{z}_{w_i}[1]\right) = \frac{\boldsymbol{z}_{w_i}[1] - \boldsymbol{z}_{w_i}[2]}{\left(\boldsymbol{z}_{w_i}[3] - \boldsymbol{z}_{w_i}[1]\right)\left(\boldsymbol{z}_{w_i}[3] - \boldsymbol{z}_{w_i}[2]\right)}. \tag{A.18}$$

Appendix B. Update formulas of $\boldsymbol{G}$

Appendix B.1. An implicit smoothness objective

Update $\boldsymbol{W} \to \boldsymbol{W}^+$

The factor $\boldsymbol{W}$ is to be updated from the objective

$$\arg\min_{\boldsymbol{W}} \|\boldsymbol{J}_{(1)} - \boldsymbol{W}((\mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V})^\top\|_F^2 + \ldots + \|\boldsymbol{J}_{(1)} - \boldsymbol{W}((\mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V})^\top\|_F^2. \tag{22}$$

Regrouping the norms leads to the equivalent expression

$$\arg\min_{\boldsymbol{W}} \left\| \begin{bmatrix} \boldsymbol{J}_{(1)}^\top \\ \vdots \\ \boldsymbol{J}_{(1)}^\top \end{bmatrix} - \begin{bmatrix} (\mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V} \\ \vdots \\ (\mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V} \end{bmatrix} \boldsymbol{W}^\top \right\|_F^2. \tag{B.1}$$

An analytical update formula is then obtained by defining

$$\boldsymbol{K} \coloneqq \begin{bmatrix} (\mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V} \\ \vdots \\ (\mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G}) \odot \boldsymbol{V} \end{bmatrix}, \tag{B.2}$$

leading to the least-squares solution

$$\boldsymbol{W}^{+^\top} = \left(\boldsymbol{K}^\top \boldsymbol{K}\right)^{-1} \boldsymbol{K}^\top \begin{bmatrix} \boldsymbol{J}_{(1)}^\top \\ \vdots \\ \boldsymbol{J}_{(1)}^\top \end{bmatrix}, \tag{B.3}$$

which is solved in a numerically more stable way using the pseudoinverse

$$\boldsymbol{W}^{+^\top} = \boldsymbol{K}^\dagger \begin{bmatrix} \boldsymbol{J}_{(1)}^\top \\ \vdots \\ \boldsymbol{J}_{(1)}^\top \end{bmatrix}. \tag{B.4}$$

Update $\boldsymbol{G} \to \boldsymbol{G}^+$

Lemma 1. *Let* (27) *be the objective function from which* $\boldsymbol{G}$ *is to be updated. The claim is that* $\boldsymbol{G}$ *appears linearly in the objective such that an analytical update formula may be derived.*

$$\arg\min_{\boldsymbol{G}} \|\boldsymbol{J}_{(3)} - (\mathcal{F}_1(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top\|_F^2 + \ldots + \|\boldsymbol{J}_{(3)} - (\mathcal{F}_s(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top\|_F^2 \tag{27}$$

Proof. Consider the vectorised form of the objective function. For clarity the dependency on $\boldsymbol{V}$ is dropped, i.e. $\mathcal{F}_i(\boldsymbol{V}) \coloneqq \mathcal{F}_i$.

$$\arg\min_{\boldsymbol{G}} \|\mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) - \mathrm{vec}\left((\mathcal{F}_1 \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top\right)\|_F^2 + \ldots + \|\mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) - \mathrm{vec}\left((\mathcal{F}_s \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^\top\right)\|_F^2 \tag{B.5}$$

Denoting the Knonecker product by '$\otimes$' we may use the property, $\mathrm{vec}\left(\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}\right) = (\boldsymbol{B}^\top \otimes \boldsymbol{A})\mathrm{vec}(\boldsymbol{X})$. With $\boldsymbol{A} = \boldsymbol{I}_N$, $\boldsymbol{B} = (\boldsymbol{V} \odot \boldsymbol{W})^\top$ and $\boldsymbol{X} = (\mathcal{F}_i \circ \boldsymbol{G})$ we have that

$$\mathrm{vec}\left((\mathcal{F}_i \circ \boldsymbol{G})\left(\boldsymbol{V} \odot \boldsymbol{W}\right)^\top\right) = \left((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N\right) \mathrm{vec}\left(\mathcal{F}_i \circ \boldsymbol{G}\right). \tag{B.6}$$

Regrouping the norms and applying the property of (19), the objective is rewritten into

$$\arg\min_{\boldsymbol{G}} \left\| \begin{bmatrix} \mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) \\ \vdots \\ \mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) \end{bmatrix} - \begin{bmatrix} \left((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N\right) \mathrm{blkdiag}\left(\mathcal{F}_1\right) \\ \vdots \\ \left((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N\right) \mathrm{blkdiag}\left(\mathcal{F}_s\right) \end{bmatrix} \mathrm{vec}\left(\boldsymbol{G}\right) \right\|_F^2, \tag{B.7}$$

from which it is clear that $\mathrm{vec}\left(\boldsymbol{G}\right)$ appears linearly in the objective. □

An analytical update formula is obtained by defining

$$\boldsymbol{K} \coloneqq \begin{bmatrix} ((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N)\,\mathrm{blkdiag}\,(\mathcal{F}_1) \\ \vdots \\ ((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N)\,\mathrm{blkdiag}\,(\mathcal{F}_s) \end{bmatrix}, \tag{B.8}$$

leading to the least-squares solution

$$\mathrm{vec}\left(\boldsymbol{G}^{+}\right) = \left(\boldsymbol{K}^{\top}\boldsymbol{K}\right)^{-1}\boldsymbol{K}^{\top}\begin{bmatrix} \mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) \\ \vdots \\ \mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) \end{bmatrix}, \tag{B.9}$$

which is solved in a numerically more stable way using the pseudoinverse

$$\mathrm{vec}\left(\boldsymbol{G}^{+}\right) = \boldsymbol{K}^{\dagger}\begin{bmatrix} \mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) \\ \vdots \\ \mathrm{vec}\left(\boldsymbol{J}_{(3)}\right) \end{bmatrix}. \tag{B.10}$$

In practice $\boldsymbol{K}$ has at least r rank deficiencies. The underdetermined system results from the fact that $\boldsymbol{G}$ is estimated while shielded behind a finite difference. Every column $\boldsymbol{g}_i$ can therefore only be estimated up to an unknown constant term. The deficiencies can be removed by fixing one element of every column, e.g. setting it equal to zero. Note that no information on the constant terms can be obtained from first order derivative information (Jacobian tensor). The constant terms need to be estimated separately (Section 6).

In case a decomposition into $r > \mathrm{rank}\,(\mathcal{J})$, is required, the system of equations will remain underdetermined. In such cases the pseudoinverse ensures that the minimum norm solution is obtained. The $\mathrm{rank}\,(\mathcal{J})$ is here defined as the minimum number of terms for which the classical factorisation of (9) is exact.

Appendix B.2. An explicit smoothness objective

Update $\boldsymbol{G} \to \boldsymbol{G}^{+}$

Lemma 2. *Let* (16) *be the objective function from which* $\boldsymbol{G}$ *is to be updated. The claim is that* $\boldsymbol{G}$ *appears linearly in this joint objective such that an analytical update formula may be derived.*

$$\arg\min_{\boldsymbol{G}} \left\|\boldsymbol{J}_{(3)} - (\mathcal{F}_C(\boldsymbol{V}) \circ \boldsymbol{G})(\boldsymbol{V} \odot \boldsymbol{W})^{\top}\right\|_F^2 + \lambda\|\left(\mathcal{F}_L(\boldsymbol{V}) \circ \boldsymbol{G}\right) - \left(\mathcal{F}_R(\boldsymbol{V}) \circ \boldsymbol{G}\right)\|_F^2 \tag{16}$$

Proof. Consider the vectorised form of the objective function. For clarity the dependency on $\boldsymbol{V}$ is dropped, e.g. $\mathcal{F}_C(\boldsymbol{V}) \coloneqq \mathcal{F}_C$.

$$\arg\min_{\boldsymbol{G}} \left\|\mathrm{vec}(\boldsymbol{J}_{(3)}) - \mathrm{vec}\left(\left(\mathcal{F}_C \circ \boldsymbol{G}\right)\left(\boldsymbol{V} \odot \boldsymbol{W}\right)^{\top}\right)\right\|_F^2 + \lambda\|\mathrm{vec}\left(\mathcal{F}_L \circ \boldsymbol{G}\right) - \mathrm{vec}\left(\mathcal{F}_R \circ \boldsymbol{G}\right)\|_F^2 \tag{B.11}$$

Also here we may invoke the property of (B.6), together with (19). The objective is then rewritten into

$$\begin{aligned} \arg\min_{\boldsymbol{G}} \; & \left\|\mathrm{vec}(\boldsymbol{J}_{(3)}) - ((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N)\,\mathrm{blkdiag}\,(\mathcal{F}_C)\,\mathrm{vec}\,(\boldsymbol{G})\right\|_F^2 \\ & + \lambda\|\left(\mathrm{blkdiag}\,(\mathcal{F}_L) - \mathrm{blkdiag}\,(\mathcal{F}_R)\right)\mathrm{vec}\,(\boldsymbol{G})\|_F^2. \end{aligned} \tag{B.12}$$

Reordering both terms leads to the expression

$$\arg\min_{\boldsymbol{G}} \left\| \begin{bmatrix} \mathrm{vec}(\boldsymbol{J}_{(3)}) \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} ((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N)\,\mathrm{blkdiag}\,(\mathcal{F}_C) \\ \sqrt{\lambda}\,(\mathrm{blkdiag}\,(\mathcal{F}_L) - \mathrm{blkdiag}\,(\mathcal{F}_R)) \end{bmatrix} \mathrm{vec}(\boldsymbol{G}) \right\|_F^2, \tag{B.13}$$

where $\boldsymbol{0} \in \mathbb{R}^{rN}$ is a vector of zeros and from which it is clear that $\mathrm{vec}(\boldsymbol{G})$ enters linearly in the objective. □

An analytical update formula is then obtained by defining

$$\boldsymbol{K} \coloneqq \begin{bmatrix} ((\boldsymbol{V} \odot \boldsymbol{W}) \otimes \boldsymbol{I}_N)\,\mathrm{blkdiag}\,(\mathcal{F}_C) \\ \sqrt{\lambda}\,(\mathrm{blkdiag}\,(\mathcal{F}_L) - \mathrm{blkdiag}\,(\mathcal{F}_R)) \end{bmatrix}, \tag{B.14}$$

leading to the least-squares solution

$$\mathrm{vec}(\boldsymbol{G}^+) = \left(\boldsymbol{K}^\top \boldsymbol{K}\right)^{-1} \boldsymbol{K}^\top \begin{bmatrix} \mathrm{vec}(\boldsymbol{J}_{(3)}) \\ \boldsymbol{0} \end{bmatrix}, \tag{B.15}$$

which is solved in a numerically more stable way using the pseudoinverse

$$\mathrm{vec}(\boldsymbol{G}^+) = \boldsymbol{K}^\dagger \begin{bmatrix} \mathrm{vec}(\boldsymbol{J}_{(3)}) \\ \boldsymbol{0} \end{bmatrix}. \tag{B.16}$$

To ensure that smoothness is promoted evenly on all columns of $\boldsymbol{G}$, a normalisation is required (similar to the update of $\boldsymbol{V}$). The block diagonal filters are therefore scaled by the root-mean-squared value of their corresponding columns. Denoting $\boldsymbol{G}'_L \coloneqq \mathcal{F}_L \circ \boldsymbol{G}$ we have that

$$\mathrm{blkdiag}(\mathcal{F}_L) = \begin{bmatrix} \frac{\boldsymbol{F}_{L_1}}{\mathrm{rms}(\boldsymbol{g}'_{L_1})} & 0 & \cdots & 0 \\ 0 & \frac{\boldsymbol{F}_{L_2}}{\mathrm{rms}(\boldsymbol{g}'_{L_2})} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{\boldsymbol{F}_{L_r}}{\mathrm{rms}(\boldsymbol{g}'_{L_r})} \end{bmatrix}. \tag{B.17}$$

In an analogue way $\mathrm{blkdiag}(\mathcal{F}_R)$ is normalised.

Also in this case $\boldsymbol{K}$ has at least r rank deficiencies stemming from unknown constant terms. The deficiencies can be removed by fixing one element of every column, e.g. setting it equal to zero. The constant terms need to be estimated separately (Section 6). Remaining deficiencies, encountered for decompositions where $r > \mathrm{rank}\,(\mathcal{J})$ are mitigated by the pseudoinverse, leading to the minimum norm solution.