Spline- and Tensor-Based Signal Reconstruction: from Structure Analysis to High-Performance Algorithms to Multiplatform Implementations and Medical Applications

Inauguraldissertation

zur

Erlangung der Würde eines Doktors der Philosophie

vorgelegt der

Philosophisch-Naturwissenschaftlichen Fakultät

der Universität Basel

von

Oleksii Morozov

aus Kharkiv, Ukraine

Basel, 2015

Originaldokument gespeichert auf dem Dokumentenserver der Universität Basel
edoc.unibas.ch

Dieses Werk ist unter dem Vertrag „Creative Commons Namensnennung-Keine kommerzielle Nutzung-Keine Bearbeitung 3.0 Schweiz“ (CC BY-NC-ND 3.0 CH) lizenziert.
Die vollständige Lizenz kann unter
creativecommons.org/licenses/by-nc-nd/3.0/ch/
eingesehen werden.

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät

auf Antrag von

Prof. Dr. Thomas Vetter, Universität Basel, Fakultätsverantwortlicher
Prof. Dr. Jürg Gutknecht, ETH Zürich, Korreferent
Prof. Dr. Michael Unser, EPF Lausanne, Korreferent

Basel, den 24.06.2014

Prof. Dr. Jörg Schibler, Dekan

This work was carried out at the Computational Medicine research group of the University Hospital of Basel under the supervision of Prof. Dr. Patrick Hunziker.

Namensnennung-Keine kommerzielle Nutzung-Keine Bearbeitung 3.0 Schweiz
(CC BY-NC-ND 3.0 CH)

Sie dürfen: **Teilen** — den Inhalt kopieren, verbreiten und zugänglich machen

Unter den folgenden Bedingungen:

Namensnennung — Sie müssen den Namen des Autors/Rechteinhabers in der von ihm festgelegten Weise nennen.

Keine kommerzielle Nutzung — Sie dürfen diesen Inhalt nicht für kommerzielle Zwecke nutzen.

Keine Bearbeitung erlaubt — Sie dürfen diesen Inhalt nicht bearbeiten, abwandeln oder in anderer Weise verändern.

Wobei gilt:

- **Verzichtserklärung —** Jede der vorgenannten Bedingungen kann **aufgehoben** werden, sofern Sie die ausdrückliche Einwilligung des Rechteinhabers dazu erhalten.
- **Public Domain (gemeinfreie oder nicht-schützbare Inhalte) —** Soweit das Werk, der Inhalt oder irgendein Teil davon zur Public Domain der jeweiligen Rechtsordnung gehört, wird dieser Status von der Lizenz in keiner Weise berührt.
- **Sonstige Rechte —** Die Lizenz hat keinerlei Einfluss auf die folgenden Rechte:
    - Die Rechte, die jedermann wegen der Schranken des Urheberrechts oder aufgrund gesetzlicher Erlaubnisse zustehen (in einigen Ländern als grundsätzliche Doktrin des **fair use** bekannt);
    - Die **Persönlichkeitsrechte** des Urhebers;
    - Rechte anderer Personen, entweder am Lizenzgegenstand selber oder bezüglich seiner Verwendung, zum Beispiel für **Werbung** oder Privatsphärenschutz.
- **Hinweis —** Bei jeder Nutzung oder Verbreitung müssen Sie anderen alle Lizenzbedingungen mitteilen, die für diesen Inhalt gelten. Am einfachsten ist es, an entsprechender Stelle einen Link auf diese Seite einzubinden.

Quelle: http://creativecommons.org/licenses/by-nc-nd/3.0/ch/ Datum: 12.11.2013

Abstract

The problem of signal reconstruction is of fundamental practical value for many applications associated with the field of signal and image processing. This work considers a particular setting where the problem is formulated using a spline-based variational approach. We mainly concentrate on the following four problem-related aspects: 1). analysis of the problem structure, 2). structure-driven derivation of high-performance solving algorithms, 3). high-performance algorithm implementations and 4). translation of these results to medical applications.

The second chapter of this work presents a tensor-based abstraction for formulation and efficient solving of problems arising in the field of multidimensional data processing. In contrast to traditional matrix abstraction the proposed approach allows to formulate tensor structured problems in an explicitly multidimensional way with preservation of the underlying structure of computations that, in turn, facilitates the derivation of highly efficient solving algorithms. In addition to being a very helpful tool for our specific problem, the proposed tensor framework with its differentiating features is well suitable for implementing a tensor programming language that offers self-optimized computations of tensor expressions by semantic analysis of their terms.

The third chapter presents a practical example how the proposed tensor abstraction can be used for solving a problem of tensor B-spline-based variational reconstruction of large multidimensional images from irregularly sampled data. Based on our tensor framework we performed a detailed analysis of the problem formulation and derived highly efficient iterative solving algorithm, which offers high computational performance when implemented on computing platforms such as multi-core and GPGPU. We successfully applied the proposed approach to a real-life medical problem of ultrasound image reconstruction from a very large set of four-dimensional (3-D+time) non-uniform measurements.

The fourth chapter presents an alternative approach to the problem of variational signal reconstruction that is based on inverse recursive filtering. We revisited a B-spline-based formulation of this problem via a detailed analysis of the problem structure moving from the uniform towards non-uniform sampling settings. As a result we derived highly efficient algorithms for computing smoothing splines of any degree with an optimal choice of regularization parameter. We extended the presented approach to higher dimensions and showed how a rich variety of non-separable multidimensional smoothing spline operators and the corresponding solutions can be computed with high efficiency. We successfully applied the proposed inverse recursive filtering approach to the problem of medical Optical Coherence Tomography.

We conclude our work by presenting a high-level approach to software/hardware co-design of high-performance streaming data processing systems on FPGA. This approach allows to develop hybrid application specific system designs by combining the flexibility of multi-processor-based systems and high-performance of dedicated hardware components. A high-level programming model that is at the center of the approach along with an integrated development environment implemented based on its principles allow software and signal processing engineers who are not FPGA experts to design high-performance hardware architectures in a short time. We show some examples of how the developed framework can be efficiently used for implementation of our tensor- and spline-based algorithms.

Acknowledgments

First of all I would like to express my immense gratitude to Prof. Patrick Hunziker for giving me the opportunity to conduct my PhD work at his Computational Medicine research group, for his support, for his constructive criticism, for the excellent discussions we had, for his patience – all that helped me a lot in achieving the pursued goals and finally to complete my PhD.

Many thanks to Prof. Thomas Vetter for being my faculty representative and for his helpful support, comments and advices. I thank very much Prof. Jürg Gutknecht and Prof. Michael Unser for their work as referees, for estimation of my research and for very interesting discussions during my PhD examination. I also thank Prof. Volker Roth who kindly agreed to chair my PhD defense.

I thank the Computer System Institute of ETH Zürich led by Prof. Jürg Gutknecht, especially Dr. Felix Friedrich and Dr. Liu Ling for a very fruitful collaborative work on FPGA-based embedded system design, the results of which flowed to the fifth chapter of my thesis.

I thank Prof. Stephan Marsch and all collaborators of the Intensive Care Clinic of the University Hospital of Basel for the support and the stimulating environment.

I thank my mother, my father and my brother for their love and trust in me. A million thanks go to my wife Elena for her love, moral support and patience.

Contents

Notation

$\mathbb{Z}$	Set of integer numbers
$\mathbb{N}$	Set of natural numbers
$A, B, \ldots$	Vector spaces denoted by upper-case italic Latin symbols
$\mathbb{R}$	Euclidean space
$\mathbb{R}^d$	d-dimensional Euclidean space
$A_1 \times \cdots \times A_p$	A space of p-th order tensors (a tensor product space)
$\mathbb{R}^{d_1 \times \cdots \times d_p} \equiv \mathbb{R}^{d_1} \times \cdots \times \mathbb{R}^{d_p}$	A space of p-th order Euclidean tensors (a Euclidean tensor product space)
$a, b, \ldots, \alpha, \beta, \ldots$	Scalars; denoted by lower-case italic Latin or Greek symbols, e.g. $a \in \mathbb{R}$, $d \in \mathbb{N}$
$\mathbf{a}, \mathbf{b}, \ldots$	Vectors are denoted by lower-case bold Latin symbols, e.g. $\mathbf{a} \in \mathbb{R}^d$
$\mathbf{A}, \mathbf{B}, \ldots$	Matrices are denoted by upper-case bold Latin symbols, e.g. $\mathbf{A} \in \mathbb{R}^{d_1 \times d_2}$
$\mathcal{A}, \mathcal{B}, \ldots$	Tensors are denoted by upper-case symbols with calligraphic font, e.g. $\mathcal{A} \in \mathbb{R}$

Chapter 1

Introduction

1.1 Spline-based variational signal reconstruction

This work considers a particular setting of the generic problem of variational signal reconstruction that is based on the classical regularized least-squares formulation and aims at minimizing the following cost function:

$$J(s) = \sum_{k=1}^{K} |s(\boldsymbol{t}_k) - f_k|^2 + R_{\boldsymbol{\lambda}}(s) \tag{1.1}$$

where $s(\boldsymbol{t}), \boldsymbol{t} \in \mathbb{R}^d$ is a continuous signal to reconstruct, $\{\boldsymbol{t}_k, f_k\}$ is a set of noisy signal observations, $R_{\boldsymbol{\lambda}}$ is some regularization operator that provides a controllable (via parameters $\boldsymbol{\lambda}$) mechanism for penalization of undesired solutions. A computational solution of this problem usually requires a discretization of the unknown function $s(\boldsymbol{t})$. In this sense B-splines offer many advantages in comparison with classical bases used for function approximation such as polynomials [1], band-limited functions [2, 3], radial-basis functions [4]. Due to their locality, convolutional, multiresolution and tensor properties, B-splines allow to implement continuous signal processing in a discrete way with high efficiency [5, 6, 7]. In this work we study particular benefits of B-splines applied to the problem formulation (1.1). We mainly concentrate on the analysis of the highly organized structure of the problem formulation induced by B-spline discretization and use the obtained analysis results for generation of highly efficient solving algorithms with their translation to practical applications in medical imaging.

1.2 Main contributions of the thesis

In this work we introduced a series of important novelties and extensions to the fields of multidimensional data processing, variational signal reconstruction, spline-based signal processing and FPGA-based embedded signal processing system design with translation of the achieved results to medical applications:

- We developed a high-level tensor framework for structural analysis of multidimensional problems. In contrast to traditional matrix abstraction the proposed approach allows natural, dimensionality-preserving formulations of tensor structured problems that facilitates derivation of computationally efficient solving algorithms.

- We successfully applied our tensor framework to the solution of a large multidimensional variational signal reconstruction problem. Analysis of the multidimensional structure of the considered problem using our framework led to derivation of a highly efficient solving algorithm that permits to overcome the "curse of dimensionality" and allows to reconstruct images in higher dimensions (3 and higher) from very large sets of non-uniformly spaced measurements. The proposed algorithm can be efficiently parallelized and can be executed with high efficiency on multiple existing computing platforms. The developed algorithm was successfully applied to a real-life medical imaging problem: reconstruction of 4D (3D+time) medical ultrasound images from a very large set of irregularly sampled data.

- We revisited the problem of spline-based variational signal reconstruction using the approach of inverse recursive filtering. As a result we derived highly efficient recursive filtering algorithms for computing approximating splines of an arbitrary order that are statistically optimal in the sense of minimization of a cross validation-based cost function. We introduced a new family of approximating spline filters with improved noise discrimination characteristics. These filters are determined based on an extension of the classical smoothing spline regularization approach.

  We extended the results obtained for uniform approximating splines to the case of non-uniform spline reconstruction. This extension led to derivation of highly efficient algorithms for computing inverse spline filtering operators and the corresponding spline solutions. Finally, we extended the proposed approach to the case of multidimensional signal reconstruction. The achieved results are not limited to only B-spline based signal representation but can be used for any shift-invariant bases. The developed solving algorithms were successfully applied to a medical imaging problem of Optical Coherence Tomography (OCT) and showed significant outperformance compared with conventional techniques used in the field.

- We introduced a high-level framework for design of high-performance streaming signal processing systems on FPGA. The proposed approach allows to build complete industrial quality FPGA hardware/software systems in a short time and is easily comprehensible to software and signal processing engineers who are not experts in the field of FPGA design. The proposed approach combines the flexibility of multiprocessor systems and high performance of dedicated application specific circuits. These features make the approach well suitable for solving the signal reconstruction problem considered in this work. We presented some examples how the developed framework can be efficiently used for implementation of our tensor- and spline-based algorithms.

Chapter 2

Tensor abstractions for structure analysis of multidimensional problems

Computational problems with tensor structure, which involve large amounts of multidimensional data, arise in many fields of science and engineering [4, 8, 9, 10, 11, 12]. The standard way of dealing with such problems is not intrinsically multidimensional, and assumes reduction of problem formulation to the classical matrix formalism, which is built on two-dimensional (matrix) and one-dimensional (vector) data structures. This is achieved by a reordering of multidimensional terms from the original problem formulation, into matrices and vectors. The derived matrix formulation is then treated using well-known techniques and algorithms of matrix algebra. Finally the solution is reconstituted into its natural multidimensional form.

While this approach reduces a multidimensional problem to the well-known standard form of matrix algebra, it introduces certain limitations. The resulting formulation is no longer explicitly multidimensional, - a fact that may create difficulty in identifying and understanding important properties inherent to the particular problem [13]. Vectorization of multidimensional objects may lead to loss of spatial data coherence [14, 15], which can adversely affect the performance of solving algorithms. In many cases, the derived problem formulation does not have a straightforward, intuitive connection with the process of generating efficient solving algorithms.

Currently, a growing interest in objects with more then two dimensions – tensors, has become evident in the scientific community. Introduced in Tensor Analysis and Multilinear Algebra, tensors gained the attention of practitioners of diverse fields (see the excellent review by Kolda [12]). However, translating tensor mathematics to a convenient computational framework raises many issues [16]; including, convenient notation, ease of algorithm implementation, performance.

We proposed a comprehensive framework [17] for solving tensor structured problems that allows natural formulation of multidimensional problems using multidimensional data structure directly. The proposed framework is based on a generalization of the concepts of matrix algebra to multiple dimensions; it incorporates and unifies existing approaches from multidimensional signal processing [18, 19, 20, 21], recent developments in the field of tensor analysis [22], and multilinear algebra [23, 24, 25, 26, 12, 27, 16].

2.1 Tensors

Tensors are elements of a mathematical abstraction invented in XIX century by Italian mathematicians Ricci-Curbastro and Levi-Civita. The proposed theory later developed into Tensor Analysis and Multilinear Algebra and was successfully applied in physics such as in continuum mechanics [22] and in the theory of relativity [28]. According to the tensor theory a tensor is an abstract multidimensional entity that can be represented by a multidimensional array of its components that are defined in respect to a chosen basis and transform accordingly to a change of the basis. More formally a tensor can be defined as an element of a tensor product space that is constructed based on the outer product of a number of vector spaces (see 2.B).

2.1.1 Tensor notation

The tensor notation used in our framework is based on the original notation from Tensor Analysis with some modifications that we introduced for the sake of making the abstraction more convenient and practical for applications in the field of multidimensional data and signal processing.

As an example consider a scalar field defined on a discrete finite three-dimensional grid with extents $[N_X, N_Y, N_Z]$. To each direction of the grid we assign a vector space with the corresponding dimensionality: $X \subseteq \mathbb{R}^{N_X}, Y \subseteq \mathbb{R}^{N_Y}, Z \subseteq \mathbb{R}^{N_Z}$. In this case the scalar field can be represented as a tensor that belongs to the corresponding tensor product space $X \times Y \times Z \subseteq \mathbb{R}^{N_X \times N_Y \times N_Z}$. The components of this tensor we designate as a multidimensional array with multiple indices $\mathcal{T}^{xyz}$ with direct connection of an index and its correspondent vector space (e.g. $x \leftrightarrow X$). The number of indices in the designation determines the order of the tensor, thus the tensor $\mathcal{T}$ has the order of 3.

In general, tensors indices can be either covariant (subscripts, e.g. $\mathcal{T}_{xyz}$) or contravariant (superscripts), depending on the way the tensor components transform in respect to a change of basis (see 2.A). This property of a tensor index is called variance. Tensors can contain indices of different variance (e.g. $\mathcal{T}_z^{xy}$), such tensors are called mixed.

To distinguish multiple coordinate systems defined in the same vector space we use subindices or accents such as $'$, $\tilde{}$, $\hat{}$, $\dot{}$ etc. For example, a linear map that defines a transformation from a basis $\mathbf{e}_1$ to a basis $\mathbf{e}_2$ in the same vector space X can be designated as $\mathcal{A}_{x_2}^{x_1}$ or $\mathcal{A}_x^{\hat{x}}$. Note, that in principle, the same tensor $\mathcal{A}_{x_2}^{x_1}$ can be considered as an element of the tensor product space $X_1 \times X_2$, where X_1 and X_2 are distinct vector spaces. But the two cases will be well distinguishable based on the definitions used in a specific problem context.

2.2 Tensor operations

2.2.1 Outer product

Consider three tensors $\mathcal{A}^{xy}, \mathcal{B}^{z}_{t}, \mathcal{C}_{u}$. The outer product of these tensors is expressed as

$$\mathcal{D}^{xyz}_{tu} = \mathcal{A}^{xy} \cdot \mathcal{B}^{z}_{t} \cdot \mathcal{C}_{u} \tag{2.1}$$

Thus, the outer product leads to expansion of the tensor order.

2.2.2 Contraction

For two tensors $\mathcal{A}_x$ and $\mathcal{B}^{xyz}$ the contraction is an operation that leads to reduction of tensor order and is expressed as

$$\mathcal{C}^{yz} = \mathcal{A}_x \cdot \mathcal{B}^{xyz} \tag{2.2}$$

Here for designation of the contraction, we use the Einstein convention [28] which assumes implicit summation over a pair of indices with equal designation but different variance. Without the use of this convention the expression (2.2) would have a less compact form which in case of multiple contractions can become very cumbersome:

$$\mathcal{C}^{yz} = \sum_{x} \mathcal{A}_x \mathcal{B}^{xyz} \tag{2.3}$$

2.2.3 Generalized tensor product

Consider a transformation $\mathcal{A}^{x_1 y_1 z_1}_{xy} : X \times Y \mapsto X \times Y \times Z \times Z$ applied to a tensor $\mathcal{B}^{xyz}$. The application of the transformation can be expressed as follows:

$$\mathcal{A}^{x_1 y_1 z_1}_{xy} \cdot \mathcal{B}^{xyz} = \mathcal{C}^{x_1 y_1 z z_1} \tag{2.4}$$

The expression (2.4) combines the outer product and simultaneous contraction over indices x, y. Now assume the tensor $\mathcal{A}$ can be decomposed as the outer product of three tensors:

$$\mathcal{A}^{x_1 y_1 z_1}_{xy} = \mathcal{E}^{x_1}_{x} \cdot \mathcal{F}^{y_1}_{y} \cdot \mathcal{G}^{z_1} \tag{2.5}$$

In this case the expression (2.4) will become

$$\mathcal{E}^{x_1}_{x} \cdot \mathcal{F}^{y_1}_{y} \cdot \mathcal{G}^{z_1} \cdot \mathcal{B}^{xyz} = \mathcal{C}^{x_1 y_1 z z_1} \tag{2.6}$$

Concept of ordered spaces, commutativity of tensor product

In conventional tensor formalism (and likewise in matrix formalism), the result of expression (2.6) depends on the order of multiplicands - it is not commutative. This is due to the fact that non-contracted indices are concatenated according to the order of their appearance in the expression that is used to assure a unique result of the tensor product. With operations on mixed tensors, it can be rather difficult to track the overall order of array indices in the results. Thus in some tensor literature a special notation is used, where a unique index order is defined by use of the "dot" symbol (e.g. $\mathcal{A}^{x_1y_1z_1}{}_{\cdot\,\cdot\,\cdot\,xy}$). But this brings an undesirable side-effect in significantly decreasing the readability of tensor expressions. In contrast, separable data handling operations, like filtering in signal processing, lead to the same result independently of the order in which separable transformations are applied (note, that if taking a closer look at the expression (2.6) one can recognize that the tensors $\mathcal{E}$ and $\mathcal{F}$ actually represent such separable transformations). To avoid this formal, but practically important mismatch, and to add more flexibility to handling of tensor expressions, we introduce the following constraint: in analogy to physical space, which is ordered (right hand rule), we require vector spaces used for construction of the tensor product space and their corresponding coordinate systems to have a unique predefined order. This constraint [1] leads to a fundamental new property of the framework, compared to conventional approaches for matrix and tensors: ***the tensor product becomes commutative***, in addition to being associative. Formally this means that indices of the resulted tensor have unique positions determined by a predefined order of vector spaces. Thus complex tensor products can be evaluated in arbitrary order but lead to the same result. For example, for the ordered vector space sequence X, Y, Z we can rewrite the expression (2.6) in many ways:

$$\begin{aligned} \mathcal{E}^{x_1}_{x} \cdot \mathcal{F}^{y_1}_{z} \cdot \mathcal{G}^{z_1} \cdot \mathcal{B}^{xyz} &= \\ \mathcal{F}^{y_1}_{z} \cdot \mathcal{E}^{x_1}_{x} \cdot \mathcal{G}^{z_1} \cdot \mathcal{B}^{xyz} &= \\ \mathcal{G}^{z_1} \cdot \mathcal{E}^{x_1}_{x} \cdot \mathcal{F}^{y_1}_{z} \cdot \mathcal{B}^{xyz} &= \\ \mathcal{G}^{z_1} \cdot \mathcal{F}^{y_1}_{z} \cdot \mathcal{E}^{x_1}_{x} \cdot \mathcal{B}^{xyz} &= \mathcal{C}^{x_1y_1zz_1} \end{aligned} \tag{2.7}$$

The commutativity of the tensor multiplication, achieved hereby, considerably increases the ease of handling tensor expressions. One of the important practical values of this property consists in simplification of semantic analysis of tensor expressions performed for the purpose of reduction of computational and storage requirements. For example, the expression $\mathcal{A}^{z_1t}_{z} \cdot \mathcal{B}^{y_1}_{y} \cdot \mathcal{C}^{x_1}_{x} \cdot \mathcal{D}^{xyz}_{t}, (X \subseteq \mathbb{R}^{N_X}, Y \subseteq \mathbb{R}^{N_Y}, Z \subseteq \mathbb{R}^{N_Z}, T \subseteq \mathbb{R}^{N_T}, N_X < N_Y < N_Z < N_T, N_X N_Y > N_T)$ in our framework can be computed by reordering $\mathcal{C}^{x_1}_{x} \cdot \left(\mathcal{B}^{y_1}_{y} \cdot (\mathcal{A}^{z_1t}_{z} \cdot \mathcal{D}^{xyz}_{t})\right)$. This gives the best computational performance with minimal memory requirements for storing intermediate results.

Extension of the tensor notation

According to Einstein notation it is not permitted to have more than one index with same designation and variance. However, we observed that by allowing this, with subject to some consistency rules, we can achieve an important extension of the notation. As an example, consider the following expression

$$\mathcal{A}^{i}_{x} \cdot \mathcal{B}^{i}_{y} = \mathcal{H}^{i}_{xy} \tag{2.8}$$

[1]Note that this constraint does not limit the expressiveness of tensors, as the order of vector spaces can be changed if necessary

This operation is a tensor analogue of the Khatri-Rao product [29] which is a matching elementwise (over i) Kronecker product of two sets of vectors. Note that two formally equivalent indices are merged into a single one, without dependency on outer product or contraction applied to other indices. But the constraint is, that indices can be contracted only once after all possible merges resulting in a single pair of superscript and subscript. Here is an example which represents the ***most general form of the tensor product***, combining outer product, contraction and elementwise product

$$\mathcal{A}_x^i \cdot \mathcal{D}_i^x \cdot \mathcal{B}_y^i \cdot \mathcal{P}_{iz} \cdot \mathcal{W}_i = \left(\mathcal{A}_x^i \cdot \mathcal{B}_y^i\right) \cdot \left(\mathcal{D}_i^x \cdot \mathcal{P}_{iz} \cdot \mathcal{W}_i\right) = \mathcal{H}_{xy}^i \cdot \mathcal{N}_{iz}^x = \mathcal{R}_{yz} \tag{2.9}$$

Notice how neatly the proposed notation unifies different types of products which exist separately in Matrix Algebra (Matrix product, Kronecker product, Khatri-Rao product).

2.2.4 Addition, subtraction and multiplication by a scalar

Linearity of tensor product spaces implies definition of the following elementwise tensor operations such as tensor addition, subtraction and multiplication by a scalar:

$$\mathcal{A}_{xyz}^{x_1y_1z_1} + \mathcal{B}_{xyz}^{x_1y_1z_1} = \mathcal{C}_{xyz}^{x_1y_1z_1} \tag{2.10}$$

$$\mathcal{A}_{xyz}^{x_1y_1z_1} - \mathcal{B}_{xyz}^{x_1y_1z_1} = \mathcal{D}_{xyz}^{x_1y_1z_1} \tag{2.11}$$

$$\lambda \cdot \mathcal{A}_{xyz}^{x_1y_1z_1} = \mathcal{E}_{xyz}^{x_1y_1z_1} \tag{2.12}$$

With the introduced constraint of ordered vector spaces all properties of these operations such as associativity and commutativity remain unchanged.

2.2.5 Identity transformation - Kronecker delta

Kronecker delta defined by (2.41) is a second order tensor which is equivalent to the identity transformation. Mixed version of the tensor applied to components of a contravariant or covariant first order tensor (vector) does not introduce changes:

$$\delta_x^{x_1} \cdot \mathcal{T}^x = \mathcal{T}^{x_1} \equiv \mathcal{T}^x \tag{2.13}$$

$$\delta_x^{x_1} \cdot \mathcal{U}_{x_1} = \mathcal{U}_x \equiv \mathcal{U}_{x_1} \tag{2.14}$$

where indices x and x_1 correspond to identical coordinate systems. Covariant or contravariant versions of the Kronecker delta change the variance of tensor indices, but do not change the values of tensor components

$$\delta_{x_1x} \cdot \mathcal{T}^x = \mathcal{T}_{x_1} \equiv \mathcal{T}_x \tag{2.15}$$

$$\delta^{x_1x} \cdot \mathcal{U}_{x_1} = \mathcal{U}^x \equiv \mathcal{U}^{x_1} \tag{2.16}$$

In the same way Kronecker delta can be applied along some dimension of a higher order tensor

$$\delta_x^{x_1} \cdot \mathcal{T}^{xyz} \equiv \delta_y^{y_1} \cdot \mathcal{T}^{xyz} \cdot \delta_z^{z_1} \cdot \mathcal{T}^{xyz} \equiv \mathcal{T}^{xyz} \tag{2.17}$$

The tensor product of Kronecker deltas for different vector spaces forms the multidimensional identity transformation

$$\delta_{xyz}^{x_1y_1z_1} = \delta_x^{x_1} \cdot \delta_y^{y_1} \cdot \delta_z^{z_1} \tag{2.18}$$

$$\delta_{xyz}^{x_1y_1z_1} \cdot \mathcal{T}^{xyz} = \mathcal{T}^{x_1y_1z_1} \equiv \mathcal{T}^{xyz} \tag{2.19}$$

Note, that in the case of Euclidean spaces (most considered case in the context of multidimensional signal processing) the Kronecker delta is equivalent to the so called metric tensor [22] that defines the measure of the distance in a space and generalizes the notion of the inner product. In a general setting when covariant or contravariant versions of the metric tensor are applied to a tensor they lead to a change of the tensor components and they lower or raise, correspondingly, the tensor indices. In Euclidean spaces such transformations do not imply any change of the tensor components and change the index variance only as exemplified by (2.15) and(2.16)). This means that for Euclidean tensor spaces there is practically no difference between covariant and contravariant indices and, without considering a concrete context, indices of an individual tensor can be freely changed from subscripts to superscripts and vice versa. However, when we consider a concrete context with a tensor expression involving multiple tensors the choice of index variance is influenced by the consistency of tensor operations. For example, the product of two Euclidean tensors $\mathcal{A} \in X \times Y \times Z$ and $\mathcal{B} \in X \times Z \times W$

$$\mathcal{C}^{yw} = \mathcal{A}_z^{xy} \cdot \mathcal{B}_x^{zw} \tag{2.20}$$

which assumes contraction over indices x and z can be written in multiple ways while all of them give the equivalent resulting tensor $\mathcal{C}$:

$$\mathcal{C}_{yw} = \mathcal{A}_{xy}^{z} \cdot \mathcal{B}_{zw}^{x} \equiv \mathcal{A}_{yz}^{x} \cdot \mathcal{B}_{xw}^{z} \equiv \mathcal{A}_{y}^{xz} \cdot \mathcal{B}_{xzw} \tag{2.21}$$

$$\mathcal{C}_{w}^{y} = \mathcal{A}_{xz}^{y} \cdot \mathcal{B}_{w}^{xz} \equiv \mathcal{A}_{z}^{xy} \cdot \mathcal{B}_{xw}^{z} \equiv \mathcal{A}_{x}^{yz} \cdot \mathcal{B}_{zw}^{x} \tag{2.22}$$

$$\mathcal{C}_{y}^{w} = \mathcal{A}_{yz}^{x} \cdot \mathcal{B}_{x}^{zw} \equiv \mathcal{A}_{xyz} \cdot \mathcal{B}^{xzw} \equiv \mathcal{A}_{y}^{xz} \cdot \mathcal{B}_{xz}^{w} \tag{2.23}$$

2.2.6 Inner product

The inner product of two third order Euclidean tensors $\mathcal{A}^{xyz}$ and $\mathcal{B}^{xyz}$ using the generalized tensor product can be expressed as

$$\langle \mathcal{A}, \mathcal{B} \rangle = \mathcal{A}^{xyz} \cdot \mathcal{B}^{xyz} \cdot \mathcal{I}_{xyz} \tag{2.24}$$

where $\mathcal{I}_{xyz}$ is a tensor with all components equal to one. Using the properties of Kronecker delta it is straightforward to verify that

$$\mathcal{I}_{xyz} = \delta_{xx_1yy_1zz_1} \cdot \delta_{xyz}^{x_1y_1z_1} \tag{2.25}$$

$$\langle \mathcal{A}, \mathcal{B} \rangle = \mathcal{A}^{xyz} \cdot \mathcal{B}^{xyz} \cdot \delta_{xx_1yy_1zz_1} \cdot \delta_{xyz}^{x_1y_1z_1} = \mathcal{A}^{x_1y_1z_1} \cdot \mathcal{B}_{x_1y_1z_1} \tag{2.26}$$

In a similar way the inner product is computed for higher order tensors.

Using the definition of the inner product the squared Frobenus norm of a tensor $\mathcal{C}$ of an arbitrary order is defined as

$$\|\mathcal{C}\|_F^2 = \langle \mathcal{C}, \mathcal{C} \rangle \tag{2.27}$$

Note, that in general case of non-Euclidean spaces the inner product computation requires the use of the metric tensor [22], which in the considered case of Euclidean spaces is equivalent to Kronecker delta.

2.2.7 Tensor transposition

In classical settings transposition of a tensor is defined by changing positions of tensor indices. But in our framework the order of indices is unique according to a predefined order of vector spaces. Thus patterns like $\mathbf{A}^T\mathbf{A}$ from matrix normal equations correspond to the tensor product with change of variance by the metric tensor that in Euclidean spaces is equivalent to Kronecker delta. As an example let us find the equivalent tensor form of $\mathbf{A}^T\mathbf{A}$ from the context of computing the squared Frobenius norm of a tensor $\mathcal{T}^x = \mathcal{A}^x_y \cdot \mathcal{B}^y$:

$$\begin{gathered}\|\mathcal{T}\|_F^2 = \mathcal{T}^x \cdot \mathcal{T}^x \cdot \mathcal{I}_x = \left(\mathcal{A}^x_y \cdot \mathcal{A}^x_{y_1} \cdot \mathcal{I}_x\right) \cdot \mathcal{B}^y \cdot \mathcal{B}^{y_1} = \\ \left(\mathcal{A}^x_y \cdot \mathcal{A}^x_{y_1} \cdot \delta_{xx_1} \cdot \delta^{x_1}_x\right) \cdot \mathcal{B}^y \cdot \mathcal{B}^{y_1} = \left(\mathcal{A}_{x_1 y} \cdot \mathcal{A}^{x_1}_{y_1}\right) \cdot \mathcal{B}^y \cdot \mathcal{B}^{y_1}\end{gathered} \tag{2.28}$$

where the term $\mathcal{A}^x_y \cdot \mathcal{A}^x_{y_1} \cdot \mathcal{I}_x \equiv \mathcal{A}_{x_1 y} \cdot \mathcal{A}^{x_1}_{y_1}$ is the exact equivalent of $\mathbf{A}^T\mathbf{A}$ when the tensor $\mathcal{A}$ is represented as a standard matrix. Note, that the coordinate system y_1, which is identical to y, was introduced to allow two independent contractions $\mathcal{A}^x_y \cdot \mathcal{B}^y$ and $\mathcal{A}^x_{y_1} \cdot \mathcal{B}^{y_1}$ producing the same result $\mathcal{T}^x$. This was necessary to make the computation of the Frobenius norm consistent.

2.2.8 Tensor equations and tensor inverse

Expression $\mathcal{A}^{x_1 y_1 z_1}_{xyz} \cdot \mathcal{U}^{xyz} = \mathcal{B}^{x_1 y_1 z_1}$ represents a relation between a multidimensional input $\mathcal{U}^{xyz}$ and a multidimensional output $\mathcal{B}^{x_1 y_1 z_1}$. This relation is determined by the transformation $\mathcal{A}^{x_1 y_1 z_1}_{xyz}$. When the input is unknown, the expression describes an inverse problem. In cases where a unique solution exists, the existence of a tensor inverse is implied – a transformation which maps $\mathcal{A}$ to the identity tensor

$$\tilde{\mathcal{A}}^{x_2 y_2 z_2}_{x_1 y_1 z_1} \cdot \mathcal{A}^{x_1 y_1 z_1}_{xyz} \cdot \mathcal{U}^{xyz} = \delta^{x_2 y_2 z_2}_{xyz} \cdot \mathcal{U}^{xyz} = \mathcal{U}^{x_2 y_2 z_2} = \tilde{\mathcal{A}}^{x_2 y_2 z_2}_{x_1 y_1 z_1} \cdot \mathcal{B}^{x_1 y_1 z_1} \tag{2.29}$$

Note that equation 2.29 can be represented in an equivalent matrix form, with a proper reshaping of tensor terms. However, the explicit use of tensors in a problem formulation can improve the visibility of the problem structure and as a consequence can facilitate the development of highly-efficient algorithms. We show that in Chapter 3, where we employ the proposed tensor framework for solving a large inverse problem of multidimensional image reconstruction.

2.3 Differentiation in respect to tensor components

When dealing with an optimization problem, differentiation in respect to the optimization parameter may be required. Consider an example of differentiating a tensor valued function of the form $f(\mathcal{U}) = \mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}^{xyz}$ in respect to components of the tensor $\mathcal{U}$. Let us assume that indices (x, y, z) and (x_1, y_1, z_1) correspond to two distinct coordinate systems in Euclidean vector spaces X, Y, Z. Now we define another coordinate system (x_2, y_2, z_2) and differentiate $f(\mathcal{U})$ in respect to the components of the tensor $\mathcal{U}$ represented in that coordinate system:

$$\frac{\partial}{\partial \mathcal{U}} f(\mathcal{U}) = \frac{\partial}{\partial \mathcal{U}^{x_2y_2z_2}} \left(\mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}^{xyz} \right) = \mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \left(\frac{\partial \mathcal{U}^{xyz}}{\partial \mathcal{U}^{x_2y_2z_2}} \right) \tag{2.30}$$

Using tensor transformation properties we can write

$$\frac{\partial \mathcal{U}^{xyz}}{\partial \mathcal{U}^{x_2y_2z_2}} = \mathcal{S}^{xyz}_{x_2y_2z_2} = (\mathcal{S}_X)^{x}_{x_2} \cdot (\mathcal{S}_Y)^{y}_{y_2} \cdot (\mathcal{S}_Z)^{z}_{z_2} \tag{2.31}$$

where tensors $\mathcal{S}_X, \mathcal{S}_Y, \mathcal{S}_Z$ represent the transformations from the coordinate system (x_2, y_2, z_2) to (x, y, z). For simplicity reasons let us assume that the coordinate systems are related by identity transformations:

$$\frac{\partial \mathcal{U}^{xyz}}{\partial \mathcal{U}^{x_2y_2z_2}} = \delta^{xyz}_{x_2y_2z_2} = \delta^{x}_{x_2} \cdot \delta^{y}_{y_2} \cdot \delta^{z}_{z_2} \tag{2.32}$$

And now using the properties of Kronecker delta we get:

$$\frac{\partial}{\partial \mathcal{U}} f(\mathcal{U}) = \mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \delta^{xyz}_{x_2y_2z_2} = \mathcal{A}^{x_1y_1z_1}_{x_2y_2z_2} \tag{2.33}$$

2.3.1 Differentiation of tensor bilinear forms

With the same settings described above let us consider differentiation of the following tensor bilinear form $\mathcal{A} : (X \times Y \times Z) \times (X \times Y \times Z) \mapsto \mathbb{R}$:

$$f(\mathcal{U}) = \mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}^{xyz} \cdot \mathcal{U}_{x_1y_1z_1} \tag{2.34}$$

According to the product rule the derivative can be computed as follows:

$$\begin{gathered} \frac{\partial}{\partial \mathcal{U}} f(\mathcal{U}) = \frac{\partial}{\partial \mathcal{U}^{x_2y_2z_2}} \mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}^{xyz} \cdot \mathcal{U}_{x_1y_1z_1} = \\ \left(\mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}_{x_1y_1z_1} \right) \cdot \delta^{xyz}_{x_2y_2z_2} + \left(\mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}^{xyz} \right) \cdot \delta_{x_1x_2y_1y_2z_1z_2} \end{gathered} \tag{2.35}$$

From (2.35) we see that the derivative of the bilinear form (2.34) can be computed by two separate contractions in respect to coordinate systems (x, y, z) and (x_1, y_1, z_1) followed by formal transformation of indices and summation of the index-matched results. In case of the symmetry of the tensor $\mathcal{A}$ i.e. when

$$\mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{B}^{xyz} \cdot \mathcal{C}_{x_1y_1z_1} = \mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{B}_{x_1y_1z_1} \cdot \mathcal{C}^{xyz} \tag{2.36}$$

the derivative of $f(\mathcal{U})$ will become

$$\frac{\partial}{\partial \mathcal{U}} f(\mathcal{U}) = 2 \left(\mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}_{x_1y_1z_1} \right) \cdot \delta^{xyz}_{x_2y_2z_2} = 2 \left(\mathcal{A}^{x_1y_1z_1}_{xyz} \cdot \mathcal{U}^{xyz} \right) \cdot \delta_{x_1x_2y_1y_2z_1z_2} \tag{2.37}$$

2.4 Automatic tensor expression analysis for high-performance implementations

Because the high level tensor formulation contains the complete tensorial structure information of a complex mathematical problem, automatic expression analysis can be performed, resulting in automated software optimization and code generation. We have experimentally verified this aspect in our tensor software library and have found that such automatic analysis is capable of optimizing code, with regard to computational and memory requirements, in both a problem-specific and hardware-specific manner. This result profits strongly from the commutativity of all tensor operations – the significant differentiating feature over standard matrix formulations which are non-commutative with respect to matrix multiplication. Some aspects of such automated code generation have also been discussed in [10]. This particular topic is a promising area for future research.

2.5 Application of the framework to large real-world problems

Our framework was successfully applied to solve a large real-world computational problem – a spline-based variational reconstruction [30] of multidimensional signals from incomplete and spatially scattered measurements. The problem was originally formulated in terms of matrix algebra. In dimensions higher than two, the matrices involved in the computations become extremely large, essentially prohibiting computation of the problem on current standard hardware. For example, for a moderate size of a four-dimensional reconstruction grid $128 \times 128 \times 128 \times 16$ one would have to deal with a matrix which is represented by 7'514'144'528 nonzero elements (in compressed block-band diagonal storage, 30 GBytes in single precision). Using our framework, we reformulated the problem in terms of tensors. The resulting tensor formulation helped to analyze the mathematical structure of the problem and to derive its decomposition in the form of a 1-rank tensor decomposition known as Canonical Decomposition (CANDECOMP) [26, 12, 27, 16]. More details about the proposed tensor formulation and solving algorithm will be provided in Chapter 3.

2.6 Discussion

We introduced a framework for structural analysis of multidimensional problems. The proposed approach leads to a natural, dimensionality-preserving formulation of tensor structured problems and facilitates the induction of computationally efficient solving algorithms. The chosen formalism based on Einstein notation together with introduced commutativity of the tensor product, makes the framework well suited for implementing a tensor programming language, offering self-optimized computations of tensor expressions by semantic analysis of their terms while avoiding explicit use of tensor indexing in the actual implementations of solving programs. This was verified by implementing our own tensor library built on the principles of the proposed framework.

The properties presented above distinguish our framework from the one used in the field of tensor decompositions [23, 24, 25, 26, 12, 27, 16], but do not limit its use for solving problems studied in that field.

Our ongoing research shows that the proposed framework is very useful for solving problems of signal processing in higher dimensions, which have inherent tensor structure.

2.A Dual vectors spaces, contravariant and covariant mechanisms of transformation

Let V be a real vector space with finite number of dimensions N_V. We designate by V^* the unique vector space of same dimensionality that is dual in respect to V. Any element $\mathbf{t}^* \in V^*$ is a linear map $V \to \mathbb{R}$

$$\langle \mathbf{t}^*, \mathbf{t} \rangle \equiv \mathbf{t}^*(\mathbf{t}) \in \mathbb{R} \tag{2.38}$$

For a given base e_v of V transformations to a new base $\hat{e}_{v_1}$ and vice versa are defined by

$$\hat{e}_{v_1} = \sum_{v=1}^{N_V} \mathcal{P}^{v}_{v_1} \cdot e_v, \; e_{v_2} = \sum_{v_1=1}^{N_V} \mathcal{S}^{v_1}_{v_2} \cdot \hat{e}_{v_1} \tag{2.39}$$

where $\mathcal{P}^{v}_{v_1}$ and $\mathcal{S}^{v_1}_{v_2}$ are direct and indirect base transformations respectively. A vector from V given by $\sum_{v=1}^{N_V} \mathcal{T}^v \cdot e_v$ according to (2.39) in a new coordinate system is represented by components

$$\hat{\mathcal{T}}^{v_1} = \sum_{v=1}^{N_V} \mathcal{S}^{v_1}_{v} \cdot \mathcal{T}^v \tag{2.40}$$

Components of such vectors which transform indirectly in respect to transformation of the base are called contravariant.

For the base e_v of V and dual base e^{v_1} of V^* the following equation is satisfied

$$\langle e^{v_1}, e_v \rangle \equiv \delta^{v_1}_{v} = \begin{cases} 1, & \text{if } v_1 = v \\ 0, & \text{if } v_1 \neq v \end{cases} \tag{2.41}$$

where $\delta^{v_1}_{v}$ is Kronecker delta. Thus the dual base e^{v_1} transforms according to

$$\hat{e}^{v_2} = \sum_{v_1=1}^{N_V} \mathcal{S}^{v_2}_{v_1} \cdot e^{v_1}, \; e^{v_3} = \sum_{v_2=1}^{N_V} \mathcal{P}^{v_3}_{v_2} \cdot \hat{e}^{v2} \tag{2.42}$$

In contrast to contravariant vector components, components of vectors from dual space V^* transform in the same way as the base of V

$$\mathcal{T}_{v_2} = \sum_{v_1=1}^{N_V} \mathcal{P}^{v_1}_{v_2} \cdot \mathcal{T}_{v_1} \tag{2.43}$$

Components of such vectors are called covariant.

2.B Generalized definition of a tensor

Definition 2.B.1. Let an ordered set of real vector spaces $U_1, \dots, U_P$, $V_1, \dots, V_Q$ with finite dimensions $I_1, \dots, I_P$, $J_1, \dots, J_Q$ and their dual vector space $U_1^*, \dots, U_P^*$, $V_1^*, \dots, V_Q^*$ be given. By identifying $(U_i^*)^*$ with U_i we define every vector $\mathbf{u}_i \in U_i$ to be a linear map $U_i^* \to \mathbb{R}$ given by

$$\mathbf{u}_i(\mathbf{u}_i^*) \equiv \langle \mathbf{u}_i^*, \mathbf{u}_i \rangle \in \mathbb{R} \tag{2.44}$$

for arbitrary $\mathbf{u}_i^* \in U_i^*$, where $\langle , \rangle$ denotes the inner product. With $(P+Q)$ vectors $\mathbf{u}_1 \in U_1, \dots, \mathbf{u}_P \in U_P$, $\mathbf{v}_1^* \in V_1^*, \dots, \mathbf{v}_Q^* \in V_Q^*$ let the element denoted $\mathcal{T} = \mathbf{u}_1 \times \cdots \times \mathbf{u}_P \times \mathbf{v}_1^* \times \cdots \times \mathbf{v}_Q^*$ be a $(P+Q)$-linear map from $U_1 \times \cdots \times U_P \times V_1^* \times \cdots \times V_Q^*$ to $\mathbb{R}$ defined by

$$\begin{aligned} \mathbf{u}_1 \times \cdots \times \mathbf{u}_P \times \mathbf{v}_1^* \times \cdots \times \mathbf{v}_Q^* \left(\mathbf{a}_1^*, \dots, \mathbf{a}_P^*, \mathbf{b}_1, \dots, \mathbf{b}_Q\right) = \\ \langle \mathbf{a}_1^*, \mathbf{u}_1 \rangle \cdots \langle \mathbf{a}_P^*, \mathbf{u}_P \rangle \langle \mathbf{v}_1^*, \mathbf{b}_1 \rangle \cdots \langle \mathbf{v}_Q^*, \mathbf{b}_Q \rangle \end{aligned} \tag{2.45}$$

where $\mathbf{a}_i^*$, $\mathbf{b}_j$ are arbitrary vectors in U_i^* and V_j respectively. The element $\mathcal{T}$ is termed a $(P+Q)$-th order decomposed tensor, P-times contravariant and Q-times covariant. The space generated by all linear combinations of decomposed tensors is termed the tensor product space. Any arbitrary tensor can be represented as a weighted sum of decomposed tensors [26, 27, 16, 31]. This definition is compatible with, and extends [24, 9].

Chapter 3

A tensor-based computational approach to large-scale iterative multidimensional signal reconstruction

In this chapter we show a practical example how the previously proposed tensor framework can be used for solving a large multidimensional problem of image reconstruction and how the analysis of this problem using the proposed tensor approach leads to derivation of highly efficient solving algorithms.

3.1 Introduction to the problem of multidimensional non-uniform signal reconstruction

The problem of data reconstruction from irregularly sampled measurements is frequently encountered in the context of biomedical imaging. For instance, the use of acquisition schemes with non-uniform, non-Cartesian spatial sampling requires a reformatting of data samples to an evenly-spaced Cartesian grid, for further analysis on a computer (e.g., ultrasound scanline-conversion). Such images are sampled sequentially along a series of scan-lines, which are non-uniformly distributed in space and in addition, do not coincide in time, further complicating the reconstruction. Similar problems of non-uniform sampling in space and/or time typically occur in many macroscopic (CT, MRI, SPECT) as well as in microscopic imaging (confocal microscopy, raster microscopy). Motion of the imaged object and time-dependent changes of image intensity also contribute to the complexity of the reconstruction problem.

Non-uniform sampling can introduce an oversampling at some locations relative to the Cartesian target grid, making an optimal smoothing method in these oversampled areas desirable, whereas in other locations of the same image, severe undersampling may occur, requiring robust interpolation. In conventional approaches, this combination of smoothing, interpolation, and time-sequential acquisition – even of a single image – are typically handled separately, facilitating implementation, but introducing multiple error sources.

The generic signal reconstruction/non-uniform interpolation problem consists of finding a continuous representation of a d-dimensional signal from arbitrarily-sampled, noisy point measurements. In the absence of any constraints on the distribution of the sampled data, the problem is ill-posed and does not have a unique solution. A natural approach for resolving this ambiguity is to introduce some prior constraints on the solution and to formulate the reconstruction as a variational problem. The cost function to be minimized is typically chosen to be the sum of two components: (1) a data term, which quantifies the fitting error between the model and the measurements, and, (2) a continuous-domain regularization functional, which introduces a penalty for non-smooth, and thus, improbable solutions. When the latter functional is quadratic, the variational problem can be solved analytically and the optimal solution is represented as a linear combination of radial basis functions (RBFs) [32, 33]. If one further imposes that the reconstruction should be independent upon any particular choice of coordinate system/spatial units, our choice of solutions narrows down to the class of thin-plate splines [34, 32], which are popular in applications, especially in the context of landmark-based image registration where the number of data points is small [35]. While thin-plate splines have many attractive mathematical properties, they are notoriously hard to deploy for large-scale interpolation problems. The main difficulty is that the underlying system matrix is dense and poorly conditioned, especially as the number of samples increases. Moreover, the evaluation of the reconstructed function at a single location is computationally expensive with a complexity that is proportional to the number of data points. During recent years, various solutions have been proposed to overcome these limitations [36, 37, 38, 39]. However, for cases of millions of data samples, the RBF approach still poses significant difficulties. To our knowledge, current RBF-based algorithms can handle up to 5'000'000 samples in 2-D and up to 250'000 samples in 3-D, running on conventional hardware [40]. There exist other approaches to the problem, which are either at risk to give unsatisfactory solutions or have limited computability similarly to the RBF [41, 42, 43].

The work of Arigovindan et al. [44, 4] presents a computationally-efficient alternative to the RBF with much better numerical behaviour. There, the use of the tensor product of 1-D B-splines as a basis for approximating the analytical solution of the optimal method was proposed. The main advantages of this approach are: the linear system arising from the formulation is well-conditioned; the system matrix is sparse and enjoys multiresolution properties that are exploited to derive an efficient Multigrid solver with a complexity mainly dependent upon the size of the reconstruction grid; the cost of resampling is minimal with a complexity $O\left((n+1)^d\right)$ per evaluated sample, where n is the degree of the applied B-spline function (for most frequent case of cubic B-spline $n = 3$), d is the number of dimensions of the reconstruction grid. The approach is extremely favorable for 2-D problems [4], but is still facing a computational bottleneck as the number of dimensions increases ("curse of dimensionality"). The intrinsic limitation is that the computational and storage requirements of the method, which are directly tied to the reconstruction grid, increase exponentially with the number of dimensions.

The authors of [45, 46] adopted the approach in [4] and presented an algorithm based on subdivision of the target grid into overlapping blocks. Sub-problems corresponding to each block were independently solved using the Multigrid solver. The proposed algorithm was successfully applied to the reconstruction of relatively large 3-D datasets. Note that in cases when a significant smoothing is imposed, the proposed block-wise solution is at risk of introducing discontinuities between neighboring blocks. In analogy to [4], the computational and storage requirements of the algorithm are still dependent linearly on the target grid size and exponentially on the number of dimensions, making it impractical for large-scale problems with dimensions greater than 3.

To overcome the computational bottleneck described above and to be able to reconstruct large, arbitrarily-sampled, multidimensional datasets, we propose to revisit the B-spline-based variational approach. In particular, we employ our tensor-based formalism introduced in Chapter 2 that reveals the intrinsic structure of the underlying system of equations, and which suggests some efficient factorization along the dimensions of the data. We then design an efficient, multilinear solver that is capable of performing the reconstruction of such data with millions of data points in any number of dimensions using standard desktop PCs in reasonable time. This makes our approach more advantageous than the original approach proposed in [4] and its adoption [46]. We evaluate the proposed algorithm on 3-D and 4-D datasets while providing some performance metrics to document its computational advantages. Finally, we present a practical example of reconstruction of a large 4-D medical ultrasound dataset from irregularly sampled noisy measurements.

3.2 Formulation of the problem

Our task is to reconstruct an unknown, continuously-defined signal $s(\boldsymbol{t}), \boldsymbol{t} \in \mathbb{R}^d$ given a set $\{f_k = s(\boldsymbol{t}_k)+n_k\}_{k=1}^{K}$ of K noisy sample values at irregularly-spaced sampling locations $\boldsymbol{t}_k$. We will do so by fitting a function, which is represented as a linear combination of basis functions, to the data according to some regularized least-squares criterion. We choose to represent our signal in a B-spline basis which combines a number of advantages: continuous representation, finite support, good approximation properties, and fast computation, as described in [5].

In contrast to the common way of RBF-based scattered data interpolation where the centers of the basis functions $\varphi(\|\boldsymbol{t} - \boldsymbol{t}_k\|)$ coincide with the sampling locations $\boldsymbol{t}_k$, the proposed uniform B-spline reconstruction is based on representing the unknown solution as a weighted sum of B-spline basis functions located at the nodes of a uniform grid, covering the domain of definition of the signal. Thus, a continuous representation of the solution in d dimensions is given by the following expansion

$$s(t_1, \dots, t_d) = \sum_{(i_1,\cdots,i_d)\in\mathbb{Z}^d} c_{i_1,\cdots,i_d}\, \beta^n(\frac{t_1}{a} - i_1)\cdots\beta^n(\frac{t_d}{a} - i_d) \tag{3.1}$$

where a is the step size of the grid which controls the quality of the discretization; $\beta^n(t)$ is the B-spline of degree n.

The cost function to be minimized combines a usual least-squares data fitting term with a continuous-domain regularization functional $\|s\|^2_{D_p}$ that penalizes non-smooth solutions; i.e.,

$$\xi(s, f) = \sum_{k=1}^{K} |s(\boldsymbol{t}_k) - f_k|^2 + \lambda \|s\|^2_{D_p} \tag{3.2}$$

where λ is a tradeoff parameter chosen for a compromise between the quality of data fit and a non-oscillating behavior of the solution; $\|s\|_{D_p}$ is Duchon's semi-norm of order p which has the important property of being scale- and rotation-invariant [34, 32]. The general d-dimensional form of this regularization functional is

$$\|s\|^2_{D_p} = \sum_{p_1+\cdots+p_d=p} \frac{p!}{p_1!\cdots p_d!} \int_{\mathbb{R}^d} \left(\frac{\partial^p s(\boldsymbol{t})}{\partial t_1^{p_1} \cdots \partial t_d^{p_d}} \right)^2 \mathrm{d}t_1 \cdots \mathrm{d}t_d. \tag{3.3}$$

It involves the sum of $N_p = \binom{p+d-1}{d-1}$ distinct quadratic terms corresponding to all partial derivatives of order p. We recall that the global, unconstrained minimization of (3.2) together with (3.3) defines the classical thin-plate spline solution [34, 32]. The main difference here is that we are searching for a solution of the form (3.1) which results in a discretized version of the problem in a uniform B-spline basis. The problem therefore boils down to finding the B-spline coefficients $c_{i_1,\cdots,i_d}$ that minimize (3.2) and therefore uniquely specify the continuous-domain reconstruction $s(\boldsymbol{t})$. Note that the discrepancy with the theoretical thin-plate spline solution can be made arbitrarily small via the adjustment of the step size a – indeed, the discretization error is guaranteed to decay like $O(a^{n+1})$ where n is the degree of the B-spline basis in (3.1) with the constraint that $n \geq p$.

3.3 Tensor structure of the problem

We will now reformulate the approximation problem making extensive use of our tensor abstraction presented in Chapter 2. The latter allows a convenient and natural description of multi-linear algebra problems and offers advantages over alternative notations [17]. For the sake of simplicity, we focus on the case $d = 3$, keeping in mind that the reasoning and formulaes are readily transposable to any number of dimensions. To each direction of the reconstruction grid with extents $[N_X, N_Y, N_Z]$, we assign a vector space with corresponding dimensionality $X \subseteq \mathbb{R}^{N_X}, Y \subseteq \mathbb{R}^{N_Y}, Z \subseteq \mathbb{R}^{N_Z}$. In conjunction to this ordered space sequence, we define the measurement vector space $F \subseteq \mathbb{R}^K$. The weighting coefficients in the expansion (3.1) are represented as a tensor in $X \times Y \times Z \subseteq \mathbb{R}^{N_X \times N_Y \times N_Z}$ with components $\mathcal{C}^{xyz}$. The evaluation of (3.1) at the location of the k-th measurement can be represented as a component of a vector in F

$$\mathcal{S}^k = \mathcal{B}^k_{xyz} \mathcal{C}^{xyz} = (\mathcal{E}^k_x \cdot \mathcal{G}^k_y \cdot \mathcal{H}^k_z) \cdot \mathcal{C}^{xyz} \tag{3.4}$$

with the convention that $\mathcal{S}^k = s(\boldsymbol{t}_k)$ and $\mathcal{C}^{xyz}$ represent the B-spline coefficients $c_{i_1,\cdots,i_d}$ in (3.1). The remaining tensors $\mathcal{E}^k_x, \mathcal{G}^k_y, \mathcal{H}^k_z$, which are the factors of $\mathcal{B}^k_{xyz}$, are obtained from the respective evaluation of the B-spline functions $\beta^n(\frac{t_{1k}}{a} - x), \beta^n(\frac{t_{2k}}{a} - y), \beta^n(\frac{t_{3k}}{a} - z)$, at the sampling locations $\boldsymbol{t}_k$. Note that these vectors are sparse due to the finite support of the B-spline basis with at most $n+1$ non-zero values.

The first quadratic term in (3.2) is then expressed as

$$\begin{aligned} \|\mathcal{S} - \mathcal{F}\|^2 = (\mathcal{S}_k - \mathcal{F}_k) \cdot \left(\mathcal{S}^k - \mathcal{F}^k\right) = \\ \mathcal{S}_k\mathcal{S}^k - 2\mathcal{S}_k\mathcal{F}^k + \mathcal{F}_k\mathcal{F}^k = \\ \left(\mathcal{C}_{x_1y_1z_1}\mathcal{B}_k^{x_1y_1z_1}\right)\left(\mathcal{B}_{xyz}^k\mathcal{C}^{xyz}\right) - 2\mathcal{C}_{xyz}\mathcal{B}_k^{xyz}\mathcal{F}^k + \mathcal{F}_k\mathcal{F}^k \end{aligned} \tag{3.5}$$

where $\mathcal{F}^k$ is the k-th entry of the measurement vector $\mathcal{F}$.

It can be shown that the regularization functional in terms of the B-spline expansion coefficients reduces to a quadratic form which has a convolutional structure [4]. In our tensor notation, it is represented by

$$\begin{aligned} \mathcal{C}_{x_1y_1z_1} \cdot \mathcal{T}_{xyz}^{x_1y_1z_1} \cdot \mathcal{C}^{xyz} = \\ \lambda\mathcal{I}^j \cdot \mathcal{C}_{x_1y_1z_1} \cdot \mathcal{P}_{jx}^{x_1} \cdot \mathcal{Q}_{jy}^{y_1} \cdot \mathcal{R}_{jz}^{z_1} \cdot \mathcal{C}^{xyz} \end{aligned} \tag{3.6}$$

where $\mathcal{I}^j \in J \subseteq \mathbb{R}^{N_p}$ is a vector with all components equal to one. In 3-D with $p = 2$, $N_p = 6$. The set of tensors $\mathcal{P}_{jx}^{x_1}, \mathcal{Q}_{jy}^{y_1}, \mathcal{R}_{jz}^{z_1}$ represent N_p d-tuples of discrete separable convolution transforms along corresponding directions of the reconstruction grid [4].

Similarly to the case of matrices and vectors, we may compute the derivatives of the cost function with respect to the components of the B-spline tensor $\mathcal{C}^{xyz}$ [17]. After setting these derivatives to zero, we end up with the following linear system of equations

$$\left(\mathcal{B}_k^{x_1y_1z_1}\mathcal{B}_{xyz}^k + \mathcal{T}_{xyz}^{x_1y_1z_1}\right) \cdot \mathcal{C}^{xyz} = \mathcal{B}_k^{x_1y_1z_1} \cdot \mathcal{F}^k \tag{3.7}$$

with the further tensor factorization

$$\mathcal{B}_k^{x_1y_1z_1}\mathcal{B}_{xyz}^k = \left(\mathcal{E}_k^{x_1} \cdot \mathcal{G}_k^{y_1} \cdot \mathcal{H}_k^{z_1}\right) \cdot \left(\mathcal{E}_x^k \cdot \mathcal{G}_y^k \cdot \mathcal{H}_z^k\right). \tag{3.8}$$

Note that the latter expression, which corresponds to the least-squares part of the cost function, involves a sum of outer products of first-order tensors. This type of decomposition is known in the field of multilinear algebra as CANonical DECOMPosition (CANDECOMP) [26, 12].

3.4 Iterative solution

At that point, we could, in principle, rely on the computational strategy proposed in [4], adapting it to the present tensor formalism for $d \geq 2$. Specifically, we might solve (3.7) by applying a Tensor Multigrid solver [17] which exploits the inherent multiresolution properties of the B-spline basis for the specification of appropriate restriction and prolongation operators [5, 4]. This allows to compute the solution with nearly linear complexity with respect to the number of grid nodes.

However, the approach proposed in [4] as well as the approach presented in [46] requires explicit computation of matrix coefficients. This may create a problem due to the fact that despite the extreme sparsity of the system tensor, the number of non-zero coefficients in higher dimensions can be so large as to exceed the storage capacities of ordinary desktop computers. For example, in four dimensions with a moderate reconstruction grid size of $128 \times 128 \times 128 \times 16$ (33'554'432 unknown B-spline coefficients), the total number of non-zero entries is 69'080'710'400 which is about 257 GByte in single precision format. By taking advantage of symmetries, the storage requirement for the system matrix can be reduced to about 30 GByte, in the best case, while requiring $\sim$ 287 GByte memory transfers, which still remains demanding.

For this reason we propose an alternative approach, based on computational tensor algebra as described in [17]. The idea is to fully exploit the tensor structure of equations (3.7) and (3.8). This approach overcomes the storage problem by efficiently recomputing the required tensor components on the fly.

3.4.1 Krylov methods and Conjugate Gradient (CG) iteration

Specifically, we propose to solve (3.7) by using a Tensor Krylov Solver (Appendix 3.A), which is based on the iterative computation of the tensor product

$$\left(\mathcal{B}_k^{x_1 y_1 z_1} \mathcal{B}_{xyz}^k + \mathcal{T}_{xyz}^{x_1 y_1 z_1}\right) \cdot \mathcal{C}^{xyz} \tag{3.9}$$

where $\mathcal{C}^{xyz}$ is the current estimate of the B-spline coefficients with index (x, y, z). The crucial point in our implementation is to compute the above tensor product, which actually represents the gradient of criterion (3.2) with respect to $\mathcal{C}$, as efficiently as possible. First, we observe that the "regularization term" $\mathcal{T}_{xyz}^{x_1 y_1 z_1} \cdot \mathcal{C}^{xyz}$ is completely specified by a set of $(N_p \times d)$ 1-D FIR filters, with impulse responses of length $h = 2n + 1$ (typ., $h = 7$ for cubic splines). The cost of the regularization filtering in the time-domain is therefore $(N_p \times d)P(2h - 1)$, taking advantage of separability, where P is the total number of the grid nodes (for 3-D: $P = N_X N_Y N_Z$). Note that, depending on the degree of the used B-spline basis, the number of filters and the size of the grid, it can be more efficient to implement the regularization filtering in the frequency domain using the Fast Fourier Transform (FFT). In 3-D and 4-D with the use of cubic B-splines the time-domain filtering offers a higher performance than FFT-based filtering, which is exploited in the current work.

To evaluate the second, "least-squares" part $\mathcal{B}_k^{x_1 y_1 z_1} \mathcal{B}_{xyz}^k \cdot \mathcal{C}^{xyz}$, we make use of (3.8) and rewrite the tensor product as

$$\left(\mathcal{E}_k^{x_1} \cdot \mathcal{E}_x^k\right) \cdot \left(\mathcal{G}_k^{y_1} \cdot \mathcal{G}_y^k\right) \cdot \left(\mathcal{H}_k^{z_1} \cdot \mathcal{H}_z^k\right) \cdot \mathcal{C}^{xyz} \tag{3.10}$$

For each k, a non-zero block of products $\left(\mathcal{E}_k^{x_1} \cdot \mathcal{E}_x^k\right), \left(\mathcal{G}_k^{y_1} \cdot \mathcal{G}_y^k\right), \left(\mathcal{H}_k^{z_1} \cdot \mathcal{H}_z^k\right)$ represents a separable transformation applied to the corresponding dimension of a small hypercube in $\mathcal{C}$. The width of the d-dimensional hypercube is $n+1$ and its bounds are dependent upon the spatial location of the given data sample. Thus, for computing (3.10), we successively loop through the K available samples, loading the corresponding k-th hypercube from $\mathcal{C}$, applying to it the set of d one-dimensional separable transforms, and finally incrementing the respective hypercube in the resulting tensor.

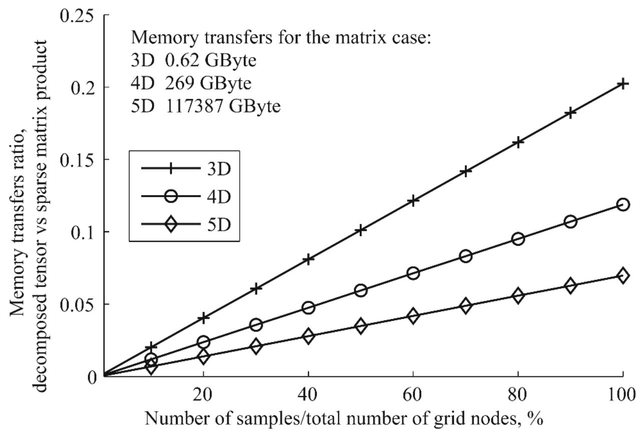
Thanks to the decomposability of (3.10), we can avoid storing the system coefficients altogether. The components of tensor factors $\mathcal{E}_x^k, \mathcal{G}_y^k, \mathcal{H}_z^k$ in the decomposition can be recomputed in each iteration at a moderate cost. For a cubic B-spline model this cost is about $10dK$ per iteration.

The advantages of the proposed, matrix-free computational scheme are the following: first, its complexity in terms of both memory transfers and arithmetic computations is linearly dependent upon the number of measurements (K), while the penalty incurred by the increase of the number of dimensions remains manageable (a factor $(n+1)^d$ instead of $\prod_{i=1}^d N_i(n+1)$ in the sparse matrix-based approach [4], where N_i are the extents of the grid); second, the processing of dense multidimensional sub-blocks provides good data locality that fits well the requirements of current hardware with cache-based architecture; third, computations can be efficiently parallelized by distributing signal measurements between multiple computational units.

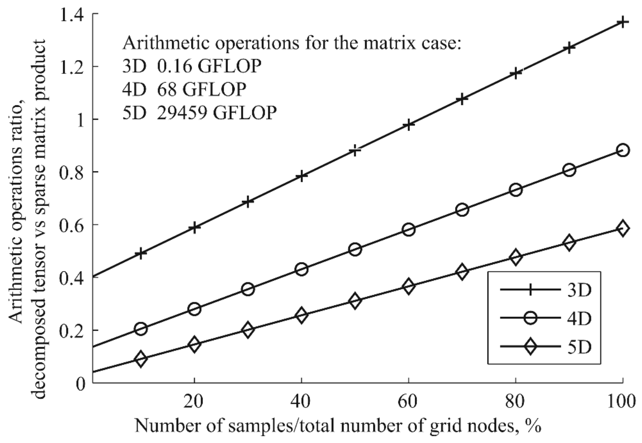
Fig. 3.1 compares the efficiency of our method to that of an explicit implementation using sparse matrix multiplication. The quantities of interest are the memory transfers and number of operations required for the evaluation of the gradient (3.9). Note that in dimensions higher than three, the benefit of the tensor method is significant even for a quite large amount of signal measurements; this is due to the weaker exponential dependency on the number of dimensions. This property of the proposed method can allow efficient reconstruction of large 3-D/3-D+time and higher dimensional data (e.g. multidimensional spectral data, physical tensor fields) encountered in recent developments in sensing technologies.

3.4.2 Multiscale initialization of the iterative solver

We did observe some degradation of the speed of convergence of the Krylov iterator when significant amounts of smoothing are required, even though the underlying linear system remains well conditioned. A typical example is shown in Fig. 3.2, which presents a result of the reconstruction of a 3-D CT data from 20% of samples with highest Laplacian values. The tradeoff factor in this experiment is 0.0034. In this particular case, we used the Tensor Conjugate Gradient iterator (TCG) (see Appendix 3.A), which was run for 20 iterations, until the relative residual norm reached 1.7×10^{-3}. Fig. 3.2(b) documents the convergence rate. In this case, the solver failed to compensate for the missing data in the regions with the lowest spatial variance, which is manifested by the presence of black spots in the reconstructed image. To overcome these difficulties, we propose to initialize the solver by providing it low frequency components of the solution. To this end, we employ a simple multiscale-based initialization. First, we obtain an approximation of the solution on the coarsest scale. Then, we interpolate the approximation to the next finer scale by means of the two scale relation filtering [5]. By successive transfer from coarse to the fine, we finally obtain the signal approximation that is used as initial condition



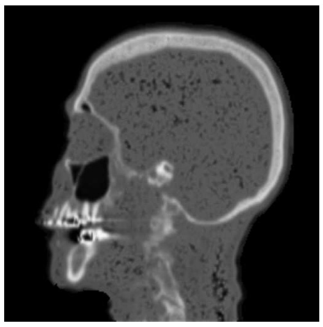
(a)



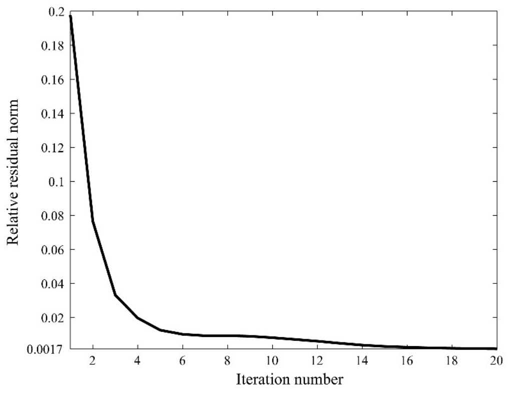
(b)

Fig. 3.1: Relative requirements of decomposed tensor vs. sparse matrix computation of the product (3.9) in single precision for memory transfers (a) and arithmetic computations (b). Reference numbers in GByte and GFLOP correspond to the sparse matrix case and do not depend on the number of data samples. Reconstruction grid has 64 nodes in all dimensions. Note that the higher the number of dimensions, the more advantageous is the use of the tensor product decomposition

for the fine-scale solver. Note that, in contrast to [4], our multiscale approach does not require explicit computation and storing of the system coefficients. Instead, we use on the fly downsampling of the sparse factors in the decompositions (3.8) and (3.6), which allows us to avoid extra storage by an additional relatively small computational cost. Our practical experience showed that the presented initialization scheme is very effective. Fig. 3.3 presents a result of the multiscale-based initialized iteration for the same data set presented above. In this case, we used 3 coarser scales for initialization: 2 iterations on the third scale, 2 iterations on the second scale and 5 iterations on the first scale. Using this initial solution, the TCG solver was run for 6 iteration until the relative residual



(a)



(b)

Fig. 3.2: Reconstruction of a 3-D data set from 20% of samples with highest Laplacian by applying non-preconditioned TCG iterator: (a) a 2-D slice taken from the reconstructed image (RMSE(%)=3.63%), (b) convergence history of the solver. Note the presence of black spots in the reconstructed image which are located in the regions with lowest spatial variance

norm reached 1.9×10^{-3} value. The resulting solution has a root mean square error $RMSE = 100 \cdot \frac{\|I_0 - I_r\|}{\|I_0\|} = 0.88\%$, where I_0 is the original image, I_r is the reconstructed image. Visually, the result of reconstruction is nearly indistinguishable from the original data.

Note that the presented multiscale initialization scheme can be used as a preconditioner for the TCG iterator. However, our practical experiments showed that this algorithm even used alone already offers good quality solutions.

3.5 Evaluation of the solving algorithm

The proposed tensor-based algorithm was implemented in MathOberon, a programming language offering multidimensional abstractions for readable, compact and efficient parallel implementations [47]. The code was tested on a PC with Intel(R) Core(TM) 2 Quad Q9400 2.67 GHz CPU equipped with 2 GByte of DDR2 800 MHz memory.

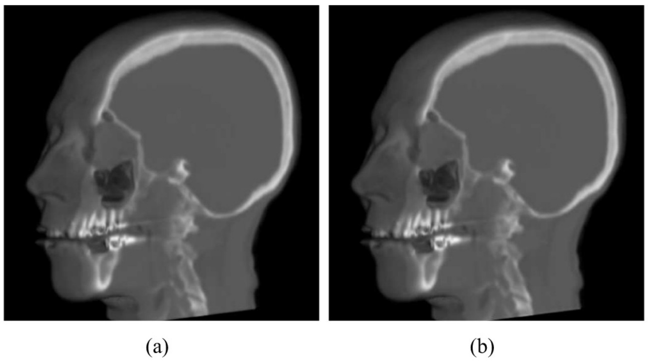


Fig. 3.3: Reconstruction of a 3-D data set from 20% of samples with highest Laplacian by applying TCG iterator with multiscale-based initialization: (a) original image, (b) reconstructed image (RMSE=0.88%)

3.5.1 Evaluation in 3-D

We evaluated the quality of reconstruction and computational performance of the proposed algorithm implementation on standard uniform 3-D datasets which are available at http://www.volvis.org. The data were sampled by taking 20% of samples with the highest values of the Laplacian, similarly to the protocol in [46]. Table 3.1 summarizes the results of the evaluation.

In these experiments, in addition to $RMSE$, we used two error measures $RMSE_1(\%) = 100 \cdot \frac{\|I_0 - I_r\|}{\sqrt{N} \cdot max(I_0)}$ and $RMSE_2(\%) = \frac{100}{max(\|\mathbf{g}_0\|)} \cdot \sqrt{\frac{\sum_{i=1}^{N} \|\mathbf{g}_i - \mathbf{g}_{0_i}\|^2}{N}}$ where N is the total number of samples, $\mathbf{g}_0$ is the gradient field computed from the original data, $\mathbf{g}$ is the gradient field computed from the reconstructed data, $\mathbf{g}_{0_i}$ and $\mathbf{g}_i$ are the values of the corresponding fields evaluated at the i-th point of the grid. All error measures are computed using all the original uniform data samples. $RMSE_1$ is used to compare our results with those presented in [46] (values in brackets); to our knowledge, this latter algorithm is the most efficient published approach for reconstructing irregular scalar data in 3-D. We used the proposed multiscale initialization scheme with 8 TCG iterations on each coarser scale, which was sufficient to get a good initialization for the finest scale iteration. For all datasets in the tests, the minimal dimension extent for coarser scales was 16 nodes, which resulted in 3 to 4 scales. On the finest scale the TCG solver was run for 10 iterations. A further increase of the number of iterations did not lead to significant improvement of the error values. As the regularizer functional we used Duchon's semi-norm of order $p = 2$. The values of the regularization tradeoff λ were chosen by minimizing a 3-fold cross validation cost individually for each dataset. Fig. 3.4 displays the behavior of the cross validation cost in an interval around the optimum for the Tooth dataset. In the performed experiments the optimum of the cross validation cost for all datasets in the tests was rather close to the optimum of the true error. For finding the location of the optimum, we used the golden section search technique; 6 to 15 evaluations of the cost function were sufficient to identify the minimum.

Fig. 3.5 presents a comparison of the behaviour of $RMSE$ and the gradient-based error

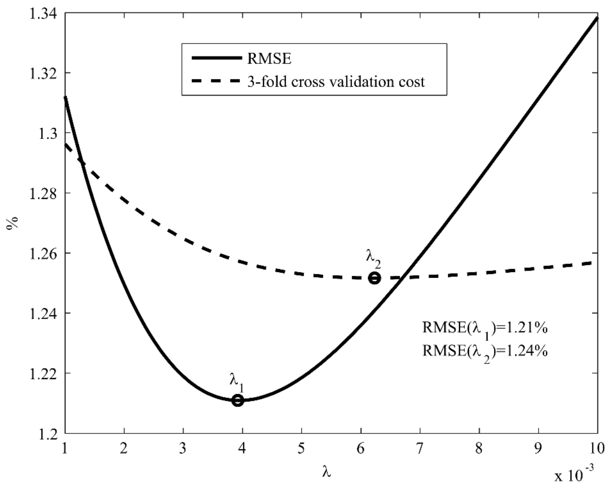


Fig. 3.4: 3-fold cross validation cost and $RMSE$ evaluated for the Tooth dataset sampled using 20% of highest Laplacian samples.

measure $RMSE_2$ with the change of the tradeoff parameter λ for the Tooth dataset. In this particular case, the optimum locations of these functions almost coincide.

Additionally, we performed tests using a regularizer of the form

$$R(s) = \lambda \int_{\mathbb{R}^d} \left(\frac{\partial^2 s(\boldsymbol{t})}{\partial t_1^2}\right)^2 + \left(\frac{\partial^2 s(\boldsymbol{t})}{\partial t_2^2}\right)^2 + \left(\frac{\partial^2 s(\boldsymbol{t})}{\partial t_3^2}\right)^2 \mathrm{d}t_1 \mathrm{d}t_2 \mathrm{d}t_3 \tag{3.11}$$

which is Duchon's semi-norm of order $p = 2$ without mixed derivative terms. In contrast to the results reported in [46], our experiments showed that the reconstruction error using (3.11) was larger or in the best case comparable to the error obtained with full Duchon's semi-norm. Fig. 3.6 presents a comparison of the reconstruction error for full Duchon's semi-norm versus the error for regularizer (3.11) both evaluated for the Tooth dataset.

The results of our 3-D evaluation indicate that our implementation is at least as good or better in terms of reconstruction quality as the block-wise Multigrid-based implementation from [45] when using a single CPU core and a comparable time budget to do a fair comparison. The execution speed of our implementation was increased by 2 with the use of 2 cores. A further increase of the number of cores did not yield an additional performance improvement; this was due to the low speed of the memory bus on our hardware. With the use of a faster memory/bus architecture, a better scalability and thus higher performance is probably achievable with the same CPU.

3.5.2 Evaluation in 4-D

One of the main advantages of our algorithm over existing approaches is the possibility of efficient reconstruction of large datasets in dimensions higher than three even when

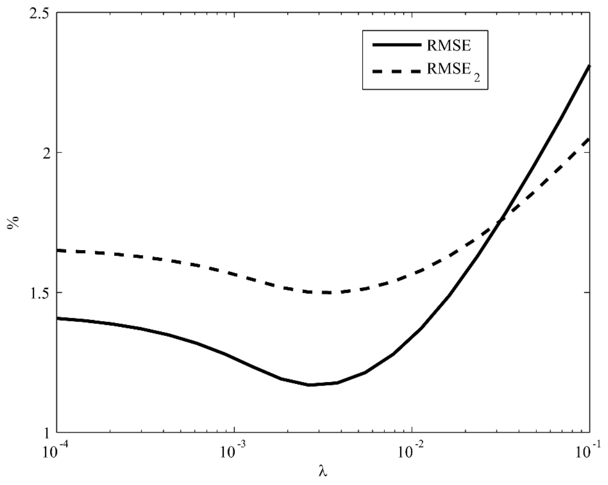


Fig. 3.5: Evaluation of $RMSE$ and the gradient-based error measure $RMSE_2$ for the Tooth dataset sampled using 20% of highest Laplacian samples.

Table 3.1: Results of the validation of the proposed algorithm on standard 3-D datasets sampled using 20% of highest Laplacian samples.

Dataset	Grid	λ	$RMSE$ %	$RMSE_1$ %	$RMSE_2$ %	Time min
Engine	$256 \times 256 \times 128$	$3.10 \cdot 10^{-3}$	2.75	0.58 (0.94)	1.40	1.12 (1.28)
Tooth	$256 \times 256 \times 160$	$6.25 \cdot 10^{-3}$	1.24	0.29 (0.18)	1.54	1.31 (1.88)
CT-Chest	$384 \times 384 \times 240$	$1.78 \cdot 10^{-3}$	1.41	0.32 (0.60)	1.09	4.38 (5.08)
Carp	$256 \times 256 \times 512$	$4.13 \cdot 10^{-4}$	1.96	0.30 (0.25)	0.34	4.81 (5.73)

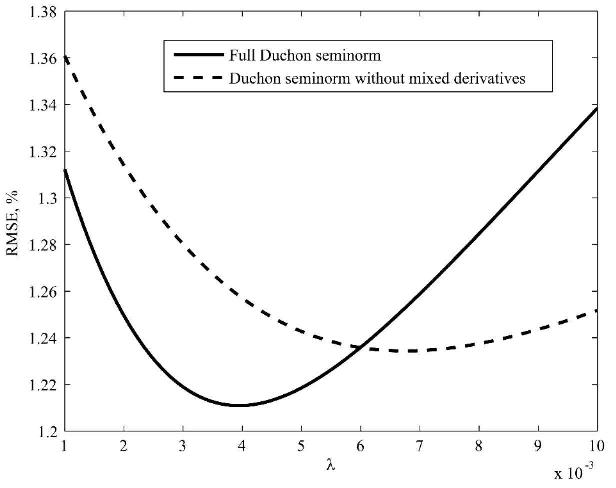
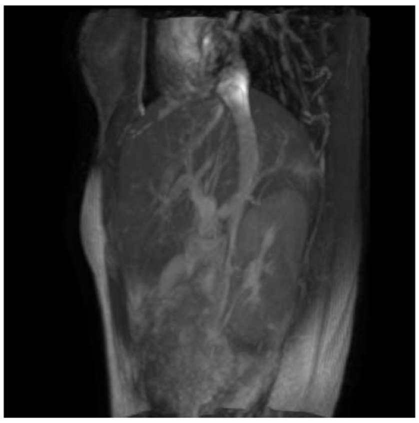
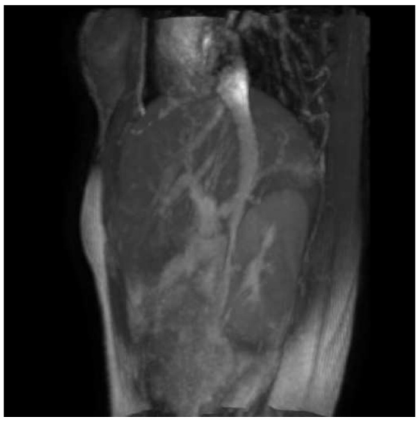


Fig. 3.6: Evaluation of $RMSE$ for full Duchon's semi-norm and the regularizer (3.11) used for the reconstruction of the Tooth dataset.



(a)



(b)

Fig. 3.7: Example of reconstruction of a 4-D MRI dataset from 30% of randomly selected samples. Presented images represent a maximum intensity projection at a single time point. (a) original data, (b) reconstructed data ($RMSE$=8.12% , $RMSE_1$=1.29%).

the computational resources are limited. We tested our algorithm implementation on a 4-D (space-time) MRI dataset kindly provided by the authors of [48]. We performed an evaluation with sampling the original data by taking 30% of random samples which for the grid size $184 \times 140 \times 30 \times 23$ corresponds to more than 5 million data points. In this experiment, we used tensor B-spline basis functions in (3.1) together with Duchon's semi-norm of order $p = 2$ and $d = 4$ as regularization functional. The tradeoff value was chosen in the same way as in 3-D tests by optimizing a 3-fold cross validation cost and was set to $6.31 \cdot 10^{-4}$. We used the proposed multiscale initialization scheme with 4 coarser scales and 8 TCG iterations on each scale. The fine-scale TCG iterator was run for 20 iterations. The reconstruction time with two CPU cores was about 5.2 minutes. Fig. 3.7 presents a 3-D visualization of a single time frame from the obtained 4-D reconstruction. Fig. 3.8 shows RMSE computed at each time point of the data. Space-time visualization with a comparison of the obtained reconstruction with the original data is available at http://www.computational.ch/downloads/mrirec4d.avi.

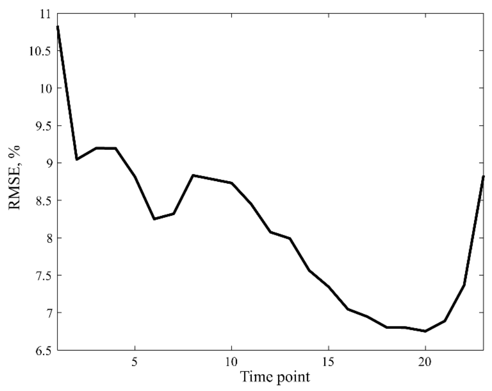


Fig. 3.8: RMSE computed at each time point of the reconstructed 4-D MRI data.

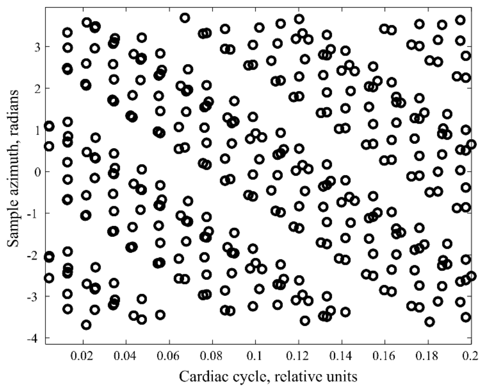


Fig. 3.9: Distribution of the azimuth versus cardiac cycle for a data set acquired by a fast continuously rotating ultrasound transducer

3.6 Application to medical imaging

We applied the proposed algorithm to a concrete imaging problem: the reconstruction of 4-D (space-time) ultrasound data acquired using a fast continuously-rotating ultrasound transducer [49, 50]. The transducer, which is based on a conventional linear array with harmonic capabilities, samples 3-D volume, while continuously rotating at the high speed of 8 rotations per second, which allows temporal resolution of about 16 volumes per second [49]. A cone-shaped volume is scanned over several seconds. Since there is no synchronization between the heart rate and the continuous rotation, the acquired data is irregularly distributed over cardiac phase and angle, as schematically shown in Fig. 3.9. Note that in this case when the data is sampled irregularly in both space and time, the ability of the reconstruction method to account for the data coherence in all dimensions is an important prerequisite for achieving good quality reconstructions. Such ability of the proposed method distinguishes it from the conventional way of frame-by-frame reconstruction.

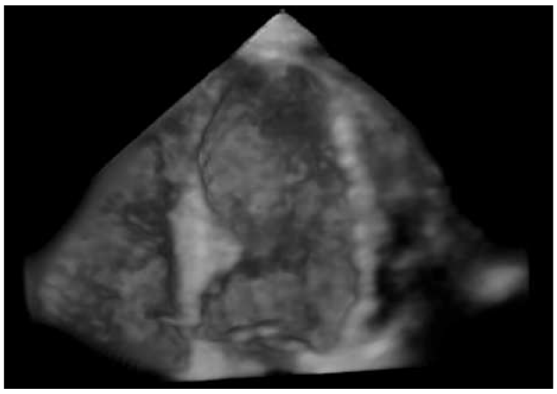


Fig. 3.10: 3-D visualization of a single time frame from the reconstruction of a 4-D arbitrarily sampled data acquired by a fast continuously rotating ultrasound transducer.

Our algorithm implementation was applied to a dataset kindly provided by the authors of [50]. This particular dataset represents the motion of the left ventricle of the heart. It consisted of more than 12 million ultrasound samples. The chosen reconstruction grid size is $128 \times 128 \times 128 \times 16$, with 16 nodes over the time axis corresponding to one cardiac cycle. The spacing of the reconstruction grid was chosen according to the sampling characteristics of the used 2-D rotating transducer. In this case, the spatial grid $128 \times 128 \times 128$ corresponds to a pixel size of about 1 mm, which is a typical axial resolution in conventional cardiac ultrasound imaging. As 4-D regularization functional, we used Duchon's semi-norm of order $p = 2$. The tradeoff value was chosen by optimizing a 3-fold cross validation cost and was set to $6.14 \cdot 10^{-2}$. For multiscale initialization, we used 4 coarser scales with 8 TCG iterations on each. On the finest scale, the TCG solver was stopped at the 30-th iteration at a relative residual norm 5.4×10^{-3}. With the use of 2 CPU cores, the total reconstruction time was about 15 minutes. Fig. 3.10 shows a 3-D visualization of a single time frame taken from the obtained reconstruction. The reconstructed data was visually assessed by experienced doctors and approved to be of a good quality. Space-time visualization of the reconstructed left ventricle dataset is available at http://www.computational.ch/downloads/echorec4d.avi.

3.7 Discussion

We presented a tensor-based approach for the efficient reconstruction of high-dimensional images from large sets of arbitrarily-sampled measurements [30]. We proposed a tensor algebra framework to analyze the structure of the B-spline reconstruction problem which led to the identification of a series of 1-D factorizations of the system tensor. We then used this representation to develop an iterative solver that is computationally and memory efficient. The critical step of our algorithm is the computation of a tensor product (update of the solution), which, due to the inherent, sparse CANDECOMP structure of the problem, can be evaluated with a complexity that is proportional to the number of measurements with a dependence on the number of physical dimensions that is significantly less than previously published solutions (reduction of the impact of the "curse of dimensionality"). The proposed algorithm can be efficiently parallelized and implemented, exploiting all currently available computing technologies such as Single Instruction Multiple Data (SIMD), Multi Core, Clusters of PCs and General Purpose GPU (GPGPU). In particular, we have built an efficient Multi-Core implementation of the proposed algorithm.

We first tested our approach on standard 3-D images using a nonuniformly subsampled portion of the data as input. We found it to be competitive with the best available Multigrid-based method [46] in the sense of providing a better or comparable reconstruction quality for a given computational budget. The proposed algorithm uses much less memory and allows computation of relatively large problems with the use of only 2 GB of memory available on our hardware. The implementation is parallel and can benefit from the use of multiple CPU cores. We also showed that the algorithm could handle more demanding tasks, such as the reconstruction of 4-D (space-time) dynamic MRI, and provide satisfactory results. In particular, we successfully applied it to a concrete large-scale imaging problem: the reconstruction of a 4-D (space-time) ultrasound signal from a large set of noisy arbitrarily sampled data acquired by a fast continuously rotating ultrasound transducer. Our ongoing work is directed towards further acceleration of the method by the use of the FPGA computing technology.

3.A Tensor Conjugate Gradient Solver

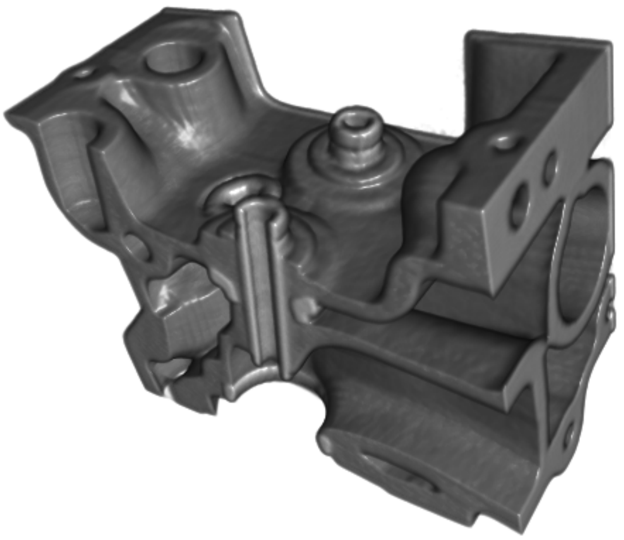
Krylov subspace solvers are among the most effective techniques for solving large linear systems of equations [51]. They operate iteratively by evaluating the residual of the system and then performing an update of the solution along an appropriate search direction. In our case, the problem reduces to solving the multi-linear system of equations (3.7) whose generic form is $\mathcal{A}_{xyz}^{x_1y_1z_1} \cdot \mathcal{C}^{xyz} = \mathcal{B}^{x_1y_1z_1}$. Since the corresponding system tensor is positive-definite, we apply a tensor variant of the Conjugate Gradient method – a special instance of a Krylov subspace solver – whose pseudo code is provided below.

```
$\mathcal{R}^{xyz} = \mathcal{B}^{xyz} - \mathcal{A}_{x_1y_1z_1}^{xyz} \cdot \mathcal{C}^{x_1y_1z_1}$        (initial residual)
$\mathcal{P}^{xyz} = \mathcal{R}^{xyz}$        (initial search directions)
$\rho = \mathcal{R}_{xyz}\mathcal{R}^{xyz}$
```

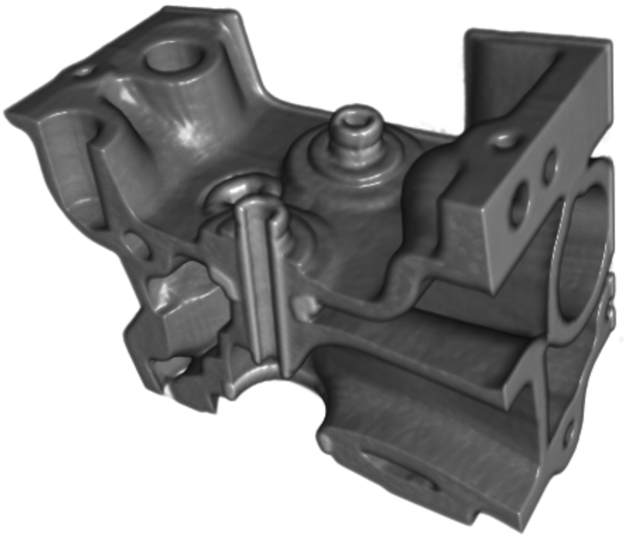
```
Repeat loop until convergence
    $\mathcal{Q}^{xyz} = \mathcal{A}^{xyz}_{x_1y_1z_1} \cdot \mathcal{P}^{x_1y_1z_1}$    (tensor computation)
    $\alpha = \frac{\rho}{(\mathcal{Q}_{xyz}\mathcal{P}^{xyz})}$
    $\mathcal{C}^{xyz} = \mathcal{C}^{xyz} + \alpha \cdot \mathcal{P}^{xyz}$    (solution update)
    $\mathcal{R}^{xyz} = \mathcal{R}^{xyz} - \alpha \cdot \mathcal{Q}^{xyz}$    (residual update)
    $\rho_1 = \mathcal{R}_{xyz}\mathcal{R}^{xyz}$
    $\mathcal{P}^{xyz} = \mathcal{R}^{xyz} + \left(\frac{\rho_1}{\rho}\right) \cdot \mathcal{P}^{xyz}$    (direction update)
    $\rho = \rho_1$
End of loop
```

Note that the computational cost of the majority of steps above is small; that, is linear in the number of unknowns (simple loop/summation over the tensor indexing variable xyz). The only step of the algorithm that is computer-intensive for large scale problems is the tensor computation that yields $\mathcal{Q}^{xyz}$. This is the part that is specific to our implementation and that is achieved according to the strategy outlined in Section 3.4.

3.B Visualization of 3-D images reconstructed during the evaluation of the proposed solving algorithm

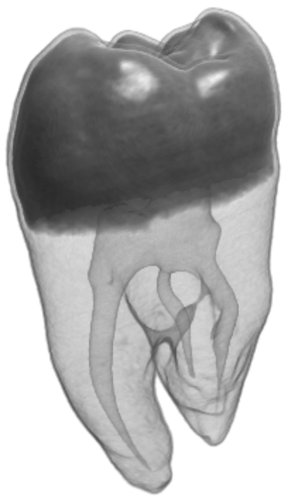


(a)

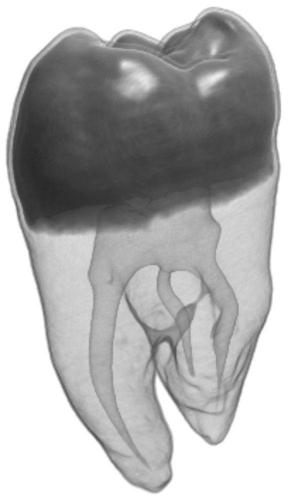


(b)

Fig. 3.11: The Engine dataset. (a) original data, (b) reconstructed data

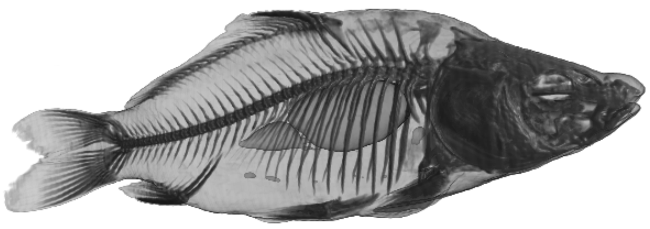


(a)

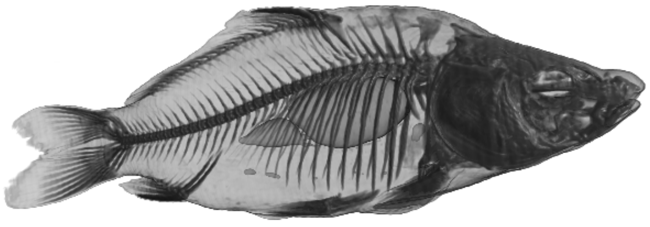


(b)

Fig. 3.12: The Tooth dataset. (a) original data, (b) reconstructed data



(a)



(b)

Fig. 3.13: The Carp dataset. (a) original data, (b) reconstructed data

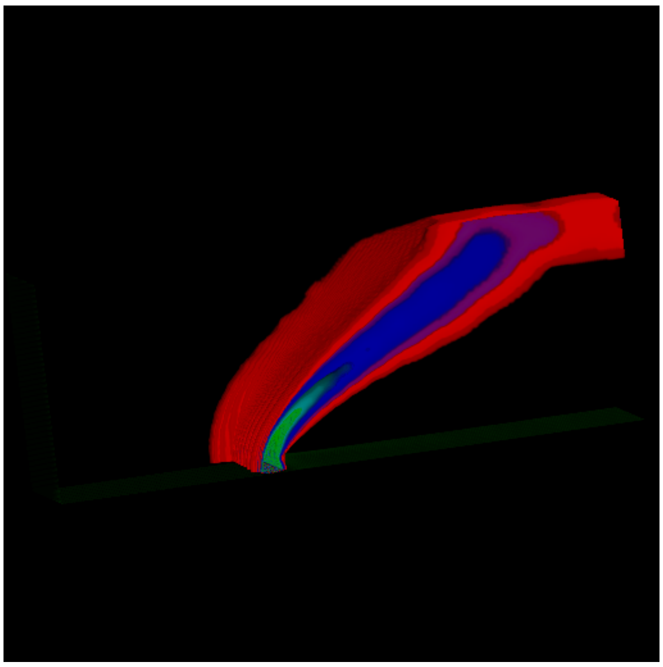


Fig. 3.14: Visualization of the reconstruction of the Blunt-Fin dataset defined on a highly anisotropic curvilinear grid. The reconstruction grid has size 186x72x50, regularization tradeoff is 0.1, RMSE1 = 0.945%

Chapter 4

Inverse spline filtering approach to signal reconstruction

In the previous chapter we presented a highly efficient iterative algorithm for solving very large multidimensional signal reconstruction problems. In this chapter we consider an alternative approach that is based on a direct computation of the spline solution via a process of inverse recursive filtering. We perform a detailed analysis of the problem by moving successively from one-dimensional uniform sampling settings towards computation of non-uniform one-dimensional and multidimensional spline solutions.

4.1 1-D uniform smoothing spline filters

We start with consideration of the one-dimensional formulation of the variational reconstruction problem. This formulation requires reconstruction of a continuous signal $s(t), t \in \mathbb{R}$ from a set of K uniformly sampled noisy measurements $\{f_k = s(t_k) + n_k, t_k = t_0 + kh\}_{k=1}^{K}$, where h is the step of the uniform sampling grid. An approach to this problem was proposed by Schoenberg [52] and Reinsch [53] and is based on solving a regularized version of the interpolation problem that aims at optimization of the following cost function:

$$J(s) = \sum_{k=1}^{K} |s(t_k) - f_k|^2 + \lambda \int_{\mathbb{R}} \left(\frac{d^p s(t)}{dt^p} \right)^2 \mathrm{d}t \tag{4.1}$$

Schoenberg [52] proved that the minimizer of the cost (4.1) is a spline of order $2p-1$ with knots at the sampling locations t_k. This spline solution is known in the literature as the smoothing or approximating spline. The classical way of computing this spline solution is based on matrix algebra and assumes solving of a linear system of equations with the system matrix having a banded Toeplitz structure [53, 54, 1]. Unser et al. [6, 7] proposed an elegant and very efficient filtering-based approach to the solution of this problem that is based on a B-spline discretization of the cost function (4.1) and does not require handling of matrices. According to this approach the signal $s(t)$ is represented using a normalized centered B-spline basis of degree n:

$$s(t) = \sum_{i \in \mathbb{Z}} c(i) \beta^n (t/h - i) \tag{4.2}$$

In practice, the infinite-long expansion (4.2) is reduced to a finite number of N coefficients with subject to a predefined boundary extension. For example, for the mirror boundary extension presented in [5] we have $N = K$.

It was shown that coefficients $c(i)$, which uniquely represent the function $s(t)$, can be computed according to the following equation represented in z-transform domain [6]:

$$C(z) = S_\lambda^{2p-1} F(z) = \frac{1}{B_1^{2p-1}(z) + \lambda \left(-z + 2 - z^{-1}\right)^p} F(z) \tag{4.3}$$

Where $C(z)$ and $F(z)$ represent the unknown discrete set of B-spline coefficients and the measurements f_k respectively, $B_1^{2p-1}(z)$ is the indirect B-spline filter of degree $2p-1$. The authors [6, 7] considered the cases of B-spline degree up to 3 that can be still treated analytically by operating with the poles of the IIR filter S_λ^{2p-1}. In the case of 1-st degree B-spline the transfer function S_λ^{2p-1} has two reciprocal real poles and, therefore, the solution can be computed in the same way as in the case of interpolation: using forward and reverse filtering with a simple 1-st order IIR filter. In case of higher B-spline degrees the solution requires a more complicated treatment because the poles of the inverse filter are generally complex and S_λ^{2p-1} cannot be decomposed into a product of simple 1-st order real IIR filters.

4.1.1 Inverse smoothing spline filter

According to the fundamental theorem of algebra every polynomial can be uniquely factored to a product of first order terms. In the case of the smoothing spline the denominator $A(z) = B_1^{2p-1}(z) + \lambda \left(-z + 2 - z^{-1}\right)^p$ of the transfer function S_λ^{2p-1} represents a symmetric FIR filter of order $2p$ that has zeroes which occur in reciprocal pairs $\{z_j, z_j^{-1}\}, |z_j| \leq 1, j = 1 \ldots p$ [7]. Moreover, due to the fact that the polynomial $A(z)$ is even and positive definite, its complex roots occur in conjugate pairs [55, 56]. So if there is a reciprocal pair $\{z_l, z_l^{-1}\}$ that is complex, then there is a corresponding reciprocal pair $\{z_l^*, \left(z_l^{-1}\right)^*\}$. The aforementioned properties make the polynomial $A(z) = a(q)z^{-q} + a(q-1)z^{-q+1} + \ldots + a(0) + \ldots + a(q-1)z^{q-1} + a(q)z^q, q = p + 1$ uniquely factorizable as:

$$A(z) = L(z)L(z^{-1}) \tag{4.4}$$

where $L(z) = l(0) - l(1)z^{-1} - l(2)z^{-2} - \ldots - l(q)z^{-q}$ is the transfer function of a real minimum-phase FIR filter that has a stable inverse $L^{-1}(z) = \frac{1}{L(z)}$. The decomposition (4.4) can be computed in multiple ways. Our interest here is to compute it as efficiently as possible using a simple algorithm that does not involve handling of matrices and complex numbers, which would be the case of direct computation of the roots using the eigenvalue decomposition of the companion matrix of the polynomial $A(z)$. Bauer [57] proposed an iterative method for computing the decomposition (4.4) that is based on Cholesky factorization applied to a semi-infinite symmetric positive definite matrix constructed from the coefficients of the polynomial $A(z)$. Bauer showed that the proposed decomposition method is convergent and self-correcting against round-off error [55]. Based on Bauer's approach we can decompose the filter $A(z)$ using the following recurrence:

$$\hat{l}(0)_u = \sqrt{a(0) - \sum_{j=1}^{p} \hat{l}(j)_u^2} \tag{4.5}$$

$$\hat{l}(v)_{u+v} = \left(a(v) - \sum_{j=1}^{p-v} \hat{l}(j)_u \hat{l}(v+j)_{u+j} \right) \frac{1}{\hat{l}(0)_u}, v = 1 \dots p, u = 0 \dots$$

The recurrent sequence $\hat{l}(v)_u$ converges to $l(v)$ in infinite number of iterations $lim_{i \to \infty} \hat{l}(v)_i = l(v)$. However, a desired accuracy (limited by the machine precision) can be achieved in a finite number of iterations N. On Fig. 4.1 we show experimentally computed dependencies of the number of iterations N required for achieving a relative l_1 approximation error $e_{l_1} \leq 1.0 \times 10^{-12}$ on the value of the regularization parameter λ for various derivative orders p with approximation error defined as

$$e_{l_1} = \frac{\sum_{k=0}^{q} |a(k) - \hat{a}(k)|}{a(0)} \tag{4.6}$$

where coefficients $\hat{a}_k$ are computed from the factorization results by polynomial multiplication (convolution) $L(z)L(z^{-1})$. From Fig. 4.1 we see that in case of small values of λ, where the impact of the regularization term $R(z) = \lambda \left(-z + 2 - z^{-1}\right)^p$ is less significant than of $B_1^{2p-1}(z)$, the algorithm can converge in 10-30 iterations. For large values of λ the filter $A(z)$ resembles the behavior of the regularization filter $R(z)$, which has zeroes that are close to the unit circle, and the number of iterations required to achieve the specified accuracy becomes larger.

An alternative way for computing the factorization (4.4) is based on the Wilson-Burg algorithm [56, 58]. The algorithm employs a Newton-Raphson iterative optimization scheme and therefore is expected to have quadratic convergence rate. However, our numerical experiments showed that when applied to factorization of the spline filter $A(z)$ the Wilson-Burg algorithm does not perform better than iteration (4.5) and in most cases requires significantly bigger number of iterations for achieving a comparable accuracy. It becomes especially slowly convergent in case of large values of λ.

An important characteristic of the transfer function $L(z)$ is that its coefficients $l(v)$ exhibit a smooth dependence on the value of the regularization parameter λ. In Fig. 4.2 we present an example how the values of $l(v)$ change over λ for the case of $p = 4$. The dependencies are plotted in the logarithmic scale. Because of presence of negative values we modified the original coefficient sequences by shifting and mirroring around the horizontal axis in a way that the minimal absolute value of the modified sequence $\tilde{l}(v)$ is same as of $l(v)$. The smooth, logarithmic scale dependencies of $\tilde{l}(v)\big(\lambda\big)$ can be approximated with high accuracy using uniform B-splines with a limited number of coefficients.

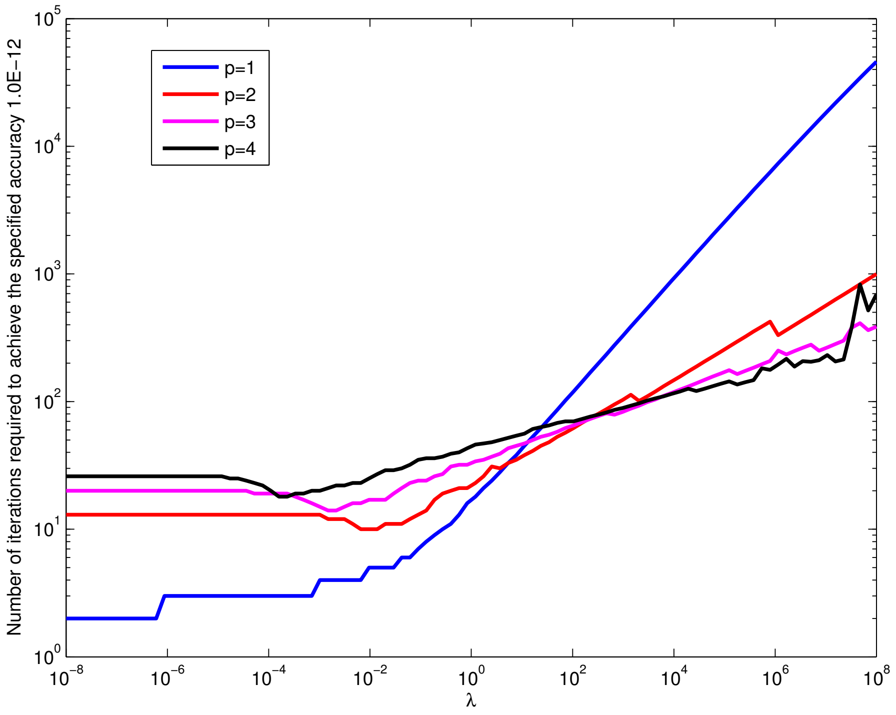


Fig. 4.1: Dependency of the number of iterations required for convergence of the recurrence (4.5) to the specified accuracy of 10^{-12} on the value of the regularization parameter λ for multiple choices of the derivative order p

Fig. 4.3 shows that for achieving the approximation error of $e_{l_1} \leq 1.0 \times 10^{-5}$ we need only 70 coefficients using a cubic B-spline discretization. Our numerical experiments with derivative orders p up to 10 showed that the above accuracy of approximation can be achieved with same 70 coefficients (which tells that the smoothness of $l(v)\big(\lambda\big)$ does not change significantly with increase of p). From the practical point of view this observation implies that for a chosen derivative order $p \leq 10$ we need only $70(p+1)$ elements of memory storage plus a few operations for evaluation of the B-spline discretized function (4 multiplications and 3 additions excluding the evaluation of cubic B-spline) to compute the inverse smoothing spline filter for a continuous range of the regularization parameter λ. This is especially important for embedded signal processing systems such as the ones based on FPGAs, where the compactness and simplicity of computations is of great importance.

Once we computed the decomposition (4.4) the B-spline coefficients $c(i)$ can be de-

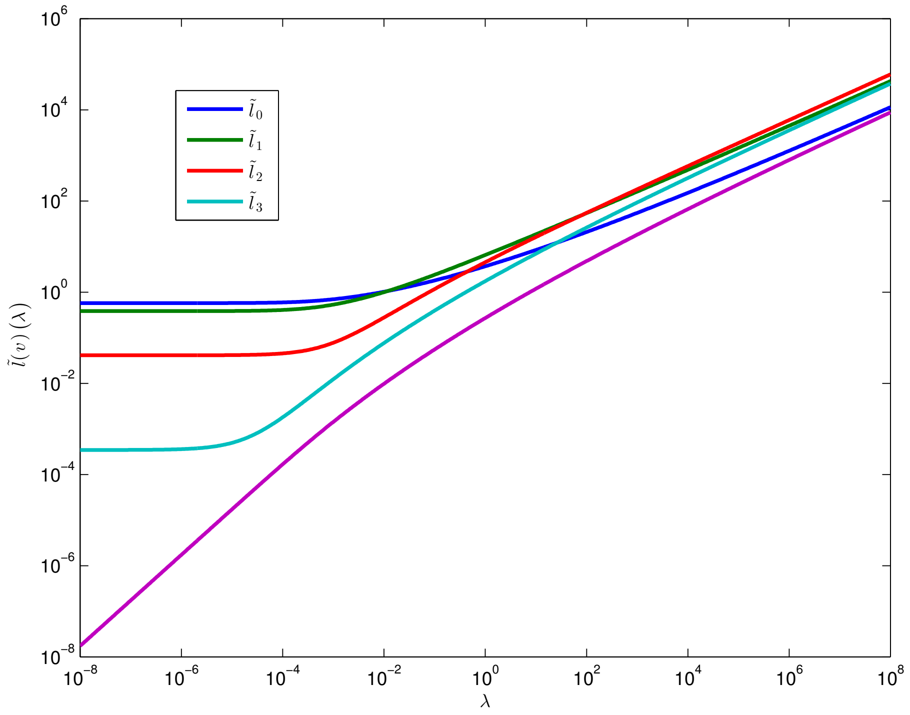


Fig. 4.2: Dependency of the coefficients of the inverse smoothing spline filter on the value of regularization parameter λ for derivative order $p = 4$

termined using the following causal and non-causal recursions:

$$\tilde{c}(i) = \frac{1}{l(0)}\left(f(i) - \sum_{v=1}^{p} l(v)\tilde{c}(i-v)\right), i = 0 \dots K-1 \tag{4.7}$$

$$c(i) = \frac{1}{l(0)}\left(\tilde{c}(i) - \sum_{v=1}^{p} l(v)c(i+v)\right), i = K-1 \dots 0 \tag{4.8}$$

This filtering algorithm requires the computation of $(4p + 2)K$ operations. Its convolutional structure makes it especially efficient for implementations on FPGA, where every operation can be performed in parallel.

4.1.2 Regularization and the choice of the tradeoff factor

The regularization parameter λ provides a mechanism for achieving a compromise between the quality of fit to the measurements $\{t_k, f_k\}$ and the smoothness of the resulting spline approximation. In general, the optimal choice of the value of λ depends on the characteristics of both signal to reconstruct and the noise contained in this signal. In some (fortunate) cases when the signal and the noise are quite well separated in the

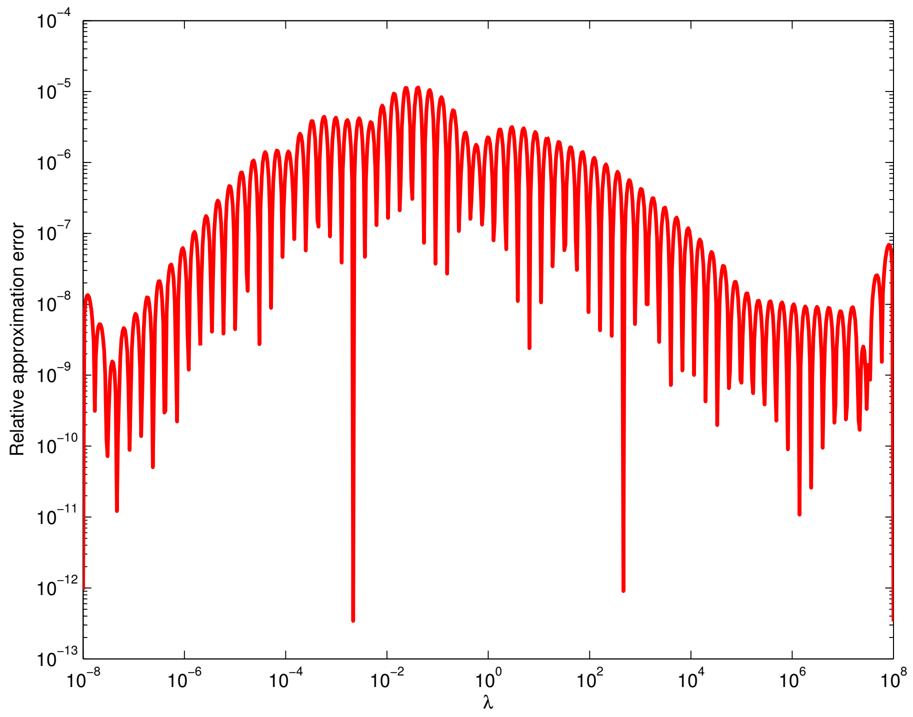


Fig. 4.3: Relative l_1 error of approximation of the inverse smoothing spline filter using uniform cubic B-spline discretization with only 70 coefficients

frequency domain the choice of the value of λ can rely on the same strategy as in conventional filtering with a low-pass filter: assure the signal frequency components are within the pass-band of the filter and achieve maximally possible attenuation of the noise in the stop-band. This choice can be accomplished based on the numerically computed dependencies presented on Fig. 4.4,4.5 ,4.6,4.7, which characterize the inverse smoothing spline filter in terms of dependency of its cut-off frequency (frequency at -3 dB amplitude level), rectangularity factor (the ratio of the cut-off frequency to the frequency at -20 dB level) and maximal attenuation within the stop-band. The presented dependencies were computed based on the transfer function of the inverse smoothing spline filter $S_{\lambda}^{2p-1}(z)$ convolved with the indirect B-spline filter $B_1^{2p-1}(z)$. From the figures one can see, that with increase of the derivative order p the smoothing spline filter approaches the ideal low-pass filter. The case described above has a practical relevance, with an exemplary application in processing of bio-medical signals such as ECG, where the uniform smoothing spline can be used for efficient implementation of template matching algorithms, see [59] for an example.

Unfortunately in most cases the spectrum of the signal does overlap with the spectrum of the noise. In these cases the best we can do is to find a value of λ that provides a balance condition when the noise is maximally suppressed while the signal is preserved as much as possible.

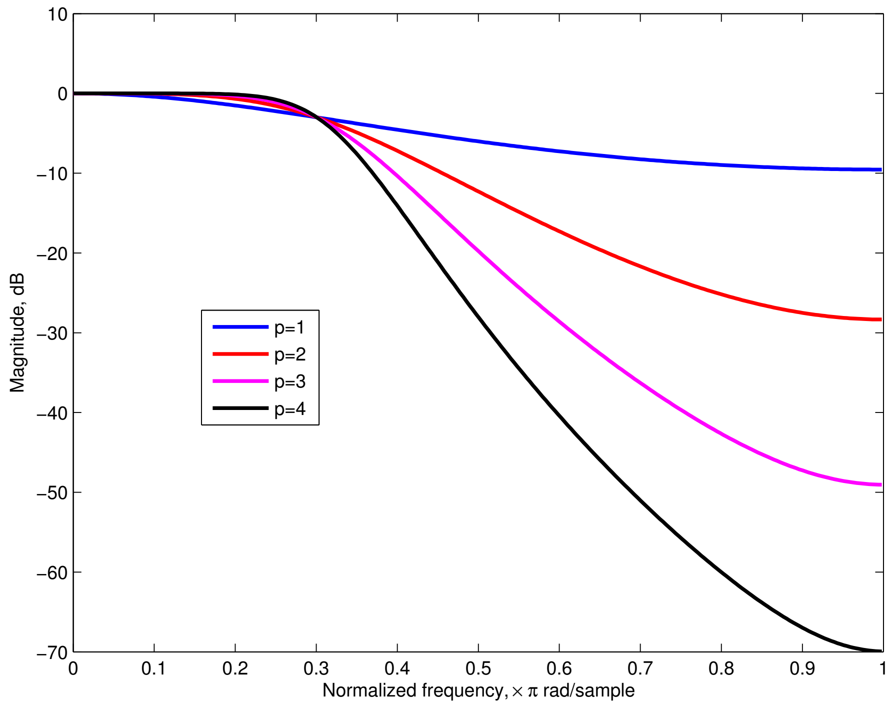


Fig. 4.4: Amplitude frequency response of the smoothing spline filter with cut-off frequency of $F_c = 0.3\ \pi$ rad/sample for multiple choices of the derivative order p

Generalized Cross-validation

Wahba et al. [60, 32] proposed to choose the value of the regularization parameter by optimizing the Generalized Cross-Validation (GCV) cost function that is based on a weighted version of the Residual Sum of Squares (RSS):

$$GCV(\lambda) = RSS \frac{K}{\left(K - tr\left(\mathbf{S}_\lambda\right)\right)^2} \tag{4.9}$$

$$RSS = \sum_{k=1}^{K} \left(s(t_k) - f_k\right)^2 \tag{4.10}$$

$\mathbf{S}_\lambda$ is a linear operator that corresponds to the filter $H(z) = B_1^{2p-1}(z) S_\lambda^{2p-1}(z)$, which transforms the sample values f_k to $s(t_k)$. The term $df = tr\left(\mathbf{S}_\lambda\right)$ is called the effective degrees of freedom of the smoothing spline and can be interpreted as the equivalent number of the spline parameters (coefficients). In the considered case of the uniform smoothing spline $df = Kh(0)$, where $h(0)$ is the coefficient of the impulse response of the filter $H(z)$ corresponding to the zero time lag. The dependency of h_0 on the value of λ is presented in Fig. 4.8. From the figure we see that with increase of λ the effective degrees of freedom decreases that confirms the fact that the corresponding smooth solution can

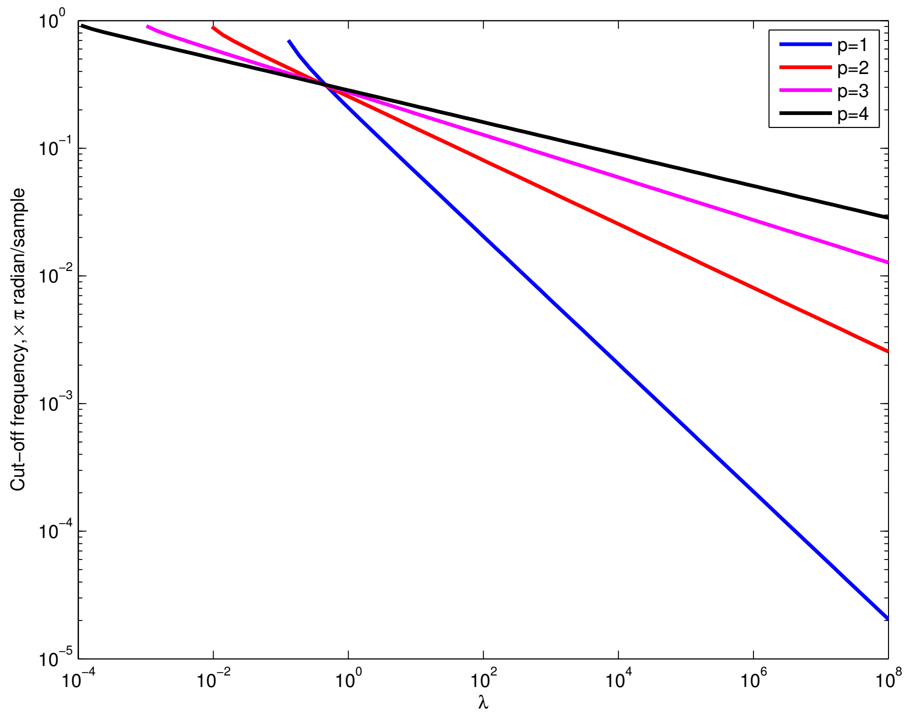


Fig. 4.5: Dependency of the cut-off frequency (-3 dB level) of the inverse smoothing spline filter on the value of the regularization parameter λ for multiple choices of the derivative order p

be equivalently approximated by a lower number of coefficients. Therefore, the GCV minimization represents an additional compromise model that aims at maximizing the data fit with minimally possible effective degrees of freedom. GCV is one of the best currently available methods for automatic choice of the value of the smoothing spline regularization parameter [61]. Note, that the GCV minimization model was derived under the assumption of the additive noise with Gaussian distribution $N(0, \sigma^2)$.

Using the ideas described in 4.1.1 we can compute the value of λ that optimizes the cost (4.9) with high efficiency and without using any matrix. The zero lag coefficient h_0 can be computed numerically by passing the Kronecker delta pulse through the filter $H(z)$: forward (4.7) and backward (4.8) recursive filtering, followed by filtering with symmetric FIR filter $B_1^{2p-1}(z)$. Similarly to the case of a B-spline-based discretization of the inverse filter it is sufficient to use 70 cubic B-spline coefficients to approximate $h_0(\lambda)$ with relative l_1 error not higher than 1.0×10^{-5}. In Fig. 4.9 we present an example of the reconstruction of a signal generated by sampling of the function $s_0(t) = sin(80t^3), t \in [0, 1]$ with number of samples $K = 1000$. The signal was contaminated by Gaussian noise with standard deviation $\sigma = 0.1$, which corresponds to the SNR of 16.6 dB. The derivative order for the regularization functional was $p = 2$. The value of λ corresponding

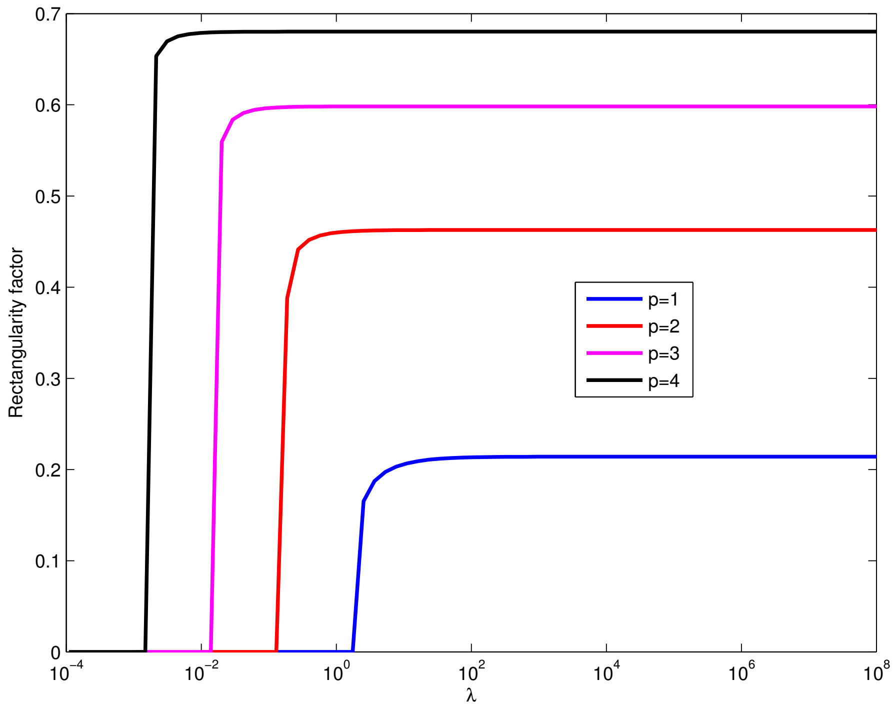


Fig. 4.6: Dependency of the rectangularity factor (the ratio of the cut-off frequency to the frequency corresponding to -20 dB level) of the inverse smoothing spline filter on the value of the regularization parameter λ for multiple choices of the derivative order p

to the minimum of the Mean Square Error $MSE = \frac{1}{\tilde{K}} \sum_{k=1}^{\tilde{K}} \left(s(\tilde{t}_k) - s_0(\tilde{t}_k)\right)^2, \tilde{t}_k = t_0 + kh/20$ estimated on a grid 20 times denser than the sampling grid t_k was $\lambda_{MSE} \approx 31$. The minimum of the GCV cost was at $\lambda_{GCV} \approx 23$. The relative Root Mean Square Error $RMSE = 100\sqrt{\frac{\sum_{k=1}^{\tilde{K}} \left(s(\tilde{t}_k) - s_0(\tilde{t}_k)\right)^2}{\sum_{k=1}^{\tilde{K}} s_0(\tilde{t}_k)^2}}$ for the smoothing spline corresponding to the minimum of MSE was $RMSE_{\lambda_{MSE}} = 5.8\%$ ($SNR \approx 24.8dB$), and for the GCV-based solution $RMSE_{\lambda_{GCV}} = 5.8\%$ ($SNR \approx 24.7dB$). The minimization of GCV cost function was performed using Matlab's *fminbnd* function that implements a minimal search algorithm based on Brent's method. Similarly to [62] instead of λ we used an intermediate variable $\sigma \in [0, 1]$ defined as:

$$\lambda = \frac{4\sigma^4}{1 - \sigma^2} \tag{4.11}$$

Fig. 4.10 shows how the behavior of the GCV cost function differs from the MSE for this particular example. From our numerical experiments with various values of σ and K we observed that the GCV solution was in a quite good correspondence to the one that minimizes MSE. As for the influence of the derivative order p in the regularization term we observed decrease of $RMSE$ with increase of p. This is confirmed by the numerical results presented in Fig. 4.11 that shows the dependencies of $RMSE_{\lambda_{MSE}}$ and $RMSE_{\lambda_{GCV}}$

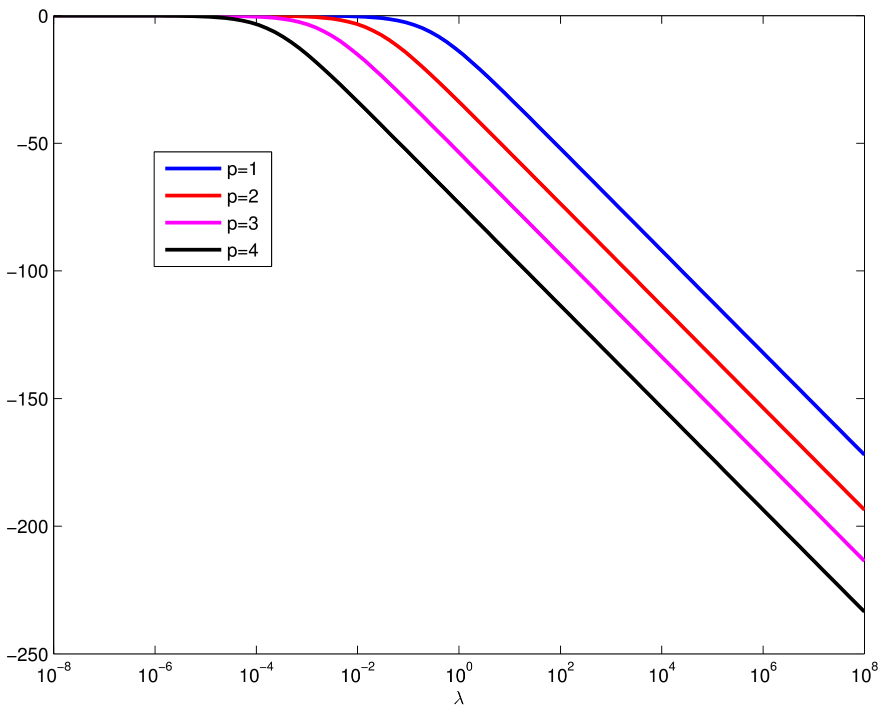


Fig. 4.7: Dependency of maximal attenuation of the inverse smoothing spline filter on the value of the regularization parameter λ for multiple choices of the derivative order p

on the value of p. These results were obtained with the same settings as the experiment described above. Every point on the plots represents the average over an ensemble of 30 realizations of the noise. The extents of the error bars are equal to doubled standard deviation of the measurements corresponding to a point. The fact that the quality of signal reconstruction improves with the increase of the derivative order p can be explained by steeper signal/noise discrimination characteristics of higher orders smoothing spline filters and their higher attenuation in the upper frequency range.

4.1.3 Regularization with multiple derivatives

In the formulation (4.1) of the smoothing spline approximation the regularization is based on penalization of the L_2 norm of p-th order derivative of the signal $s(t)$ and the space where the solution is sought for is restricted to splines of order $n = 2p - 1$. In this case we basically have two degrees of freedom that determine the spline solution: p and λ where the first parameter influences the quality of discrimination between signal and noise (steepness of the amplitude frequency response and noise attenuation level), while the second parameter controls the position of the discrimination border (cut-off frequency). When, for the purpose of improving signal/noise discrimination characteristics, we increase the derivative order p by one, this forces us to increase the complexity of the

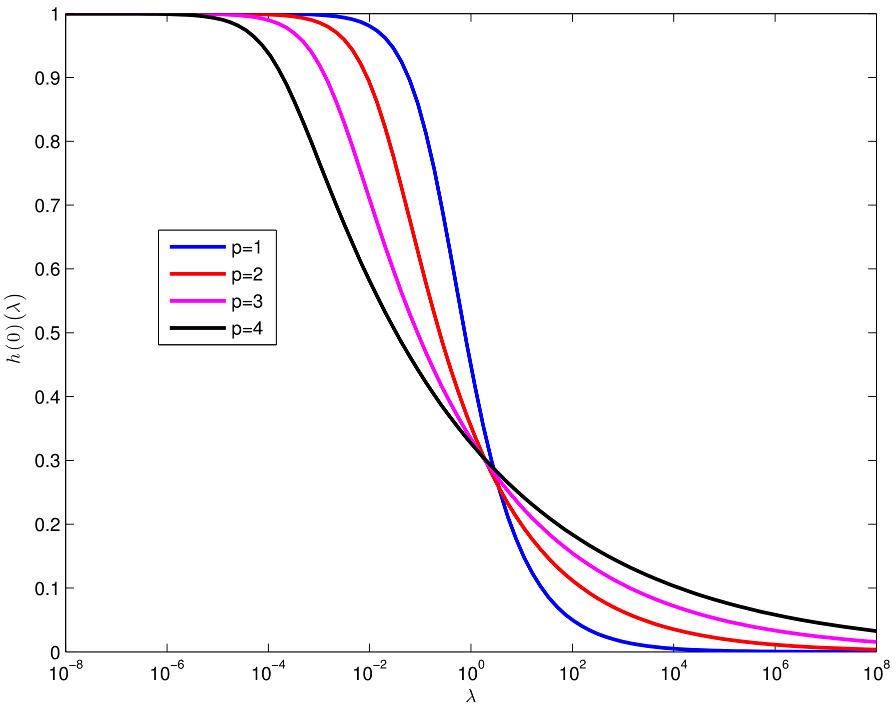


Fig. 4.8: Dependency of the zero time lag coefficient value of the filter $H(z) = B_1^{2p-1}(z)S_\lambda^{2p-1}(z)$ on the value of the regularization parameter λ for multiple choices of the derivative order p

solution by almost doubling the order of the spline. This leads us to the question, is it possible by some (preferably linear) means to stay in the space of lower order splines, while still having the possibility to control the quality of signal/noise discrimination?

Let us consider an extended form of the one-dimensional smoothing spline cost function:

$$J(s) = \sum_{k=1}^{K} |s(t_k) - f_k|^2 + \sum_{j=0}^{p} \lambda_j \int_{\mathbb{R}} \left(\frac{d^j s(t)}{dt^j} \right)^2 \mathrm{d}t, p \leq n \tag{4.12}$$

where the unknown signal $s(t)$ belongs to the space spanned by B-splines of a chosen degree n. In this case we have to solve the following equation represented in z-domain:

$$C(z) = \tilde{S}_\lambda^n(z)F(z) = \frac{B_1^n(z)}{B_1^n(z)B_1^n(z) + \sum_{j=0}^{p} \lambda_j R_j(z)} F(z) \tag{4.13}$$

where regularization filters $R_j(z)$ have impulse responses with the following structure:

$$r_j(l) = \int_{t\in\mathbb{R}} \frac{d^j}{dt^j}\beta^n(t) \frac{d^j}{dt^j}\beta^n(t-l)\,dt = (-1)^j \frac{d^{2j}\beta^{2n+1}}{dt^{2j}}(l), l = -n \dots n \tag{4.14}$$

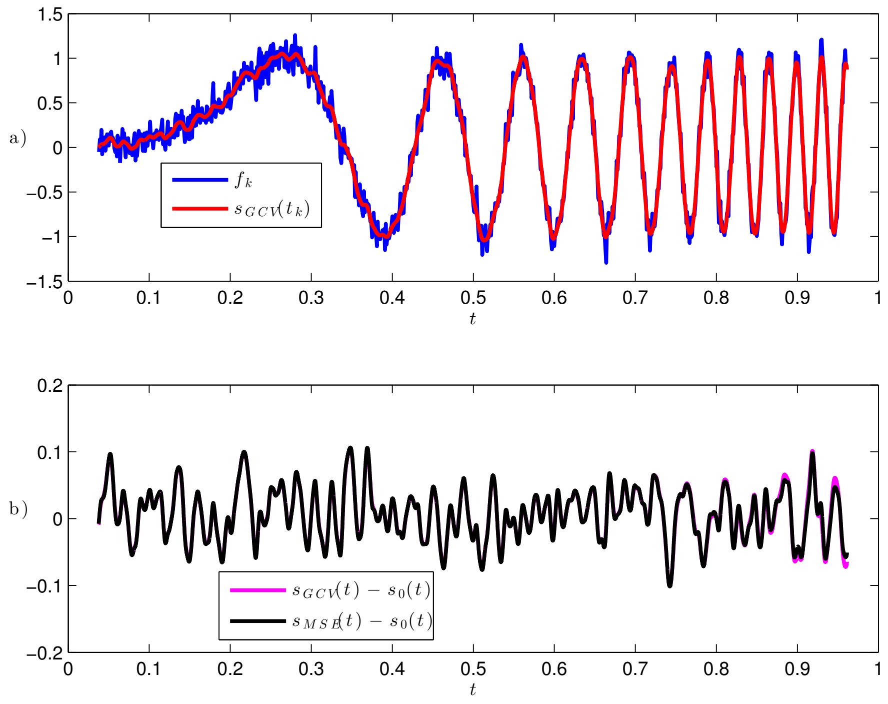


Fig. 4.9: An example of reconstruction of a signal contaminated by Gaussian noise with standard deviation of 0.1. a) Noisy signal measurements and the result of reconstruction corresponding to the minimal value of the GCV cost; b) reconstruction error computed based on GCV and MSE optimized smoothing spline solutions

Alternatively, these filters can be computed according to the following equation in z-domain:

$$R_j(z) = B_1^{2n-2j+1}(z)\Delta^j(z) \tag{4.15}$$
$$\Delta^j(z) = (-z^{-1} + 2 - z^1)^j \tag{4.16}$$

Equations (4.14) and (4.15) can be verified based on the properties of integrals and derivatives of B-spline functions [5, 6].

The solution of the considered problem can be found based on the inverse of the FIR filter in the denominator of (4.13):

$$a(l) = (b_1^n * b_1^n)(l) + \lambda_j \sum_{j=0}^{p} r_j(l), l = -n \dots n \tag{4.17}$$

where b_1^n is the impulse response of a filter obtained by sampling a B-spline of degree n [6].

To reduce the complexity of the space, where we search for the solution, we constrain the filter $a(l)$ to be positive definite, which is also the case for the classical smoothing

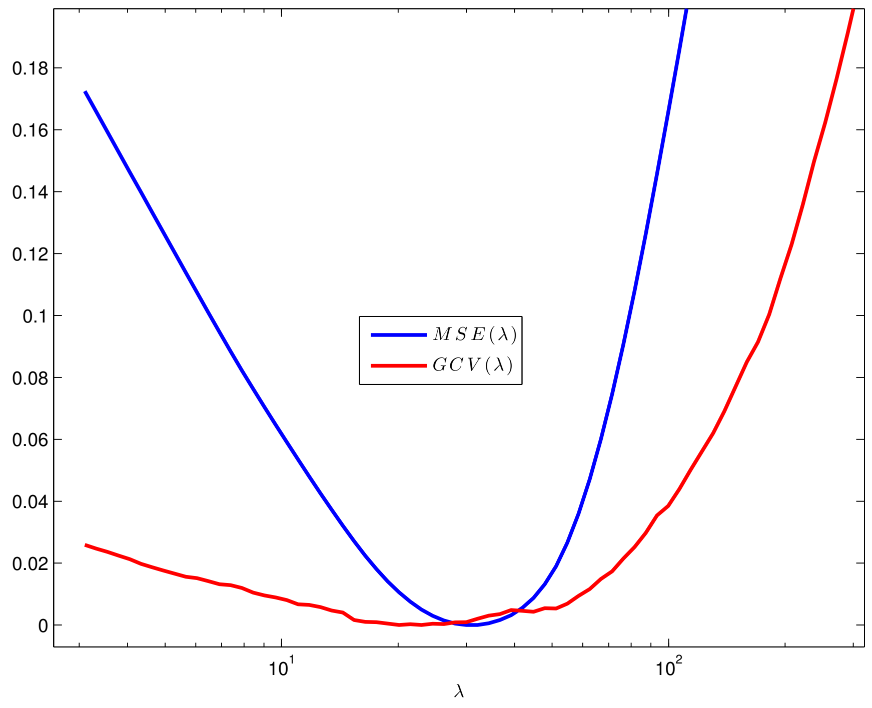


Fig. 4.10: Numerically computed dependencies of the MSE and GCV cost on the value of regularization parameter λ

spline solution with a single derivative norm. Note, that this does not necessarily mean that the weights λ_j have to be positive. Under the proposed constraint the inverse filter can be computed using the recurrence (4.5) with $p = n$.

Fig. 4.12 presents an experimental result we obtained by numerical optimization of the following weighted least squares-based cost function:

$$J(\lambda) = \sum_{k=0}^{M} w_k \left(|B_1^n(j\omega_k)\tilde{S}_\lambda^n(j\omega_k)| - \Pi_{F_c}(j\omega_k) \right)^2 \tag{4.18}$$

where F_c is the specified value of the cut-off frequency of the smoothing spline filter in $\pi\ rad/sample$, $\Pi_{F_c}(j\omega_k)$ is a real unit pulse with width F_c (in other words amplitude frequency response of an ideal low-pass filter with cut-off frequency F_c), w_k are real positive weights. For this particular example we set the weights w_k to 100 for the frequency range $[0, F_c]$ and to 1 for $(F_c, 1.0]$ to achieve higher quality of approximation within the pass-band. The number of samples M was 4096. The optimization was performed using Nelder-Mead simplex algorithm, which does not require to compute derivative of the cost function. The fact that the recurrence (4.5) breaks down in case if the filter to decompose is not positive definite was used for penalization of undesired solution candidates: every time when the argument of the square root in (4.5) is negative the value of the cost func-

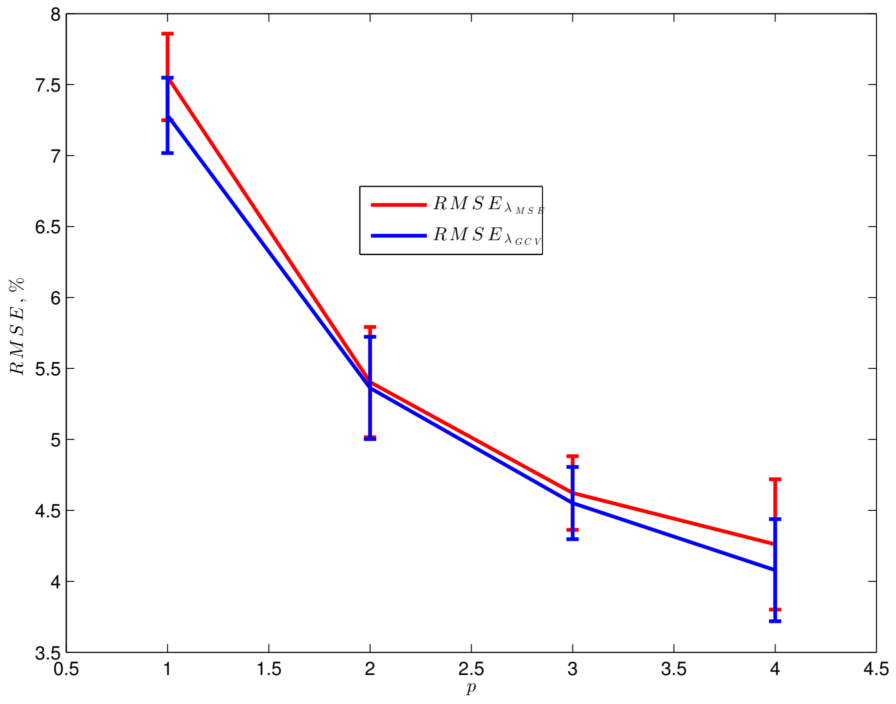


Fig. 4.11: Numerically computed dependencies of $RMSE$ for smoothing spline solutions corresponding the the minimum of MSE and the GCV cost

tion was assigned to $1/eps$. The optimization algorithm was able to converge from initial point $\lambda_j = 0$ in a few hundred of iterations. From Fig. 4.12 one can see that in terms of the steepness and rectangularity of the amplitude frequency response, the smoothing spline $\tilde{S}^3_\lambda$ performs even better than 5-th order (corresponds to $p = 3$) spline computed using the standard smoothing spline formulation. Fig. 4.13 explains the reason for the use of weights w_k: the amplitude frequency response of the spline $\tilde{S}^3_\lambda$ has ripples, similarly as in the case of Chebyshev filters of type I. The ripples can be suppressed by increasing the weights corresponding to the pass-band. It is clear that the quality of suppression of the ripples comes at the cost of decreasing the level of attenuation within the stop-band. The filters presented in Fig. 4.13 have some DC amplification. This is explained by the presence of the squared norm of the signal $\int_{\mathbb{R}} s(t)^2 \mathrm{d}t$ in the regularization term of (4.12). This can be avoided by removing the term from the formulation at the expense of some decrease in steepness of the amplitude frequency response. The fact that the numerically computed dependencies of λ_j on the value of the cut-off frequency (see Fig. 4.14) have a smooth behavior supports the notion that we either reach the global optimum or at least a well-defined local minimum.

To check the performance of the derived family of smoothing spline filters we made an experiment which is similar to the one described in section 4.1.2. For this experiment we used 10000 samples of a signal $s_0(t) = sin(800t^3), t \in [0, 1]$ contaminated by a filtered

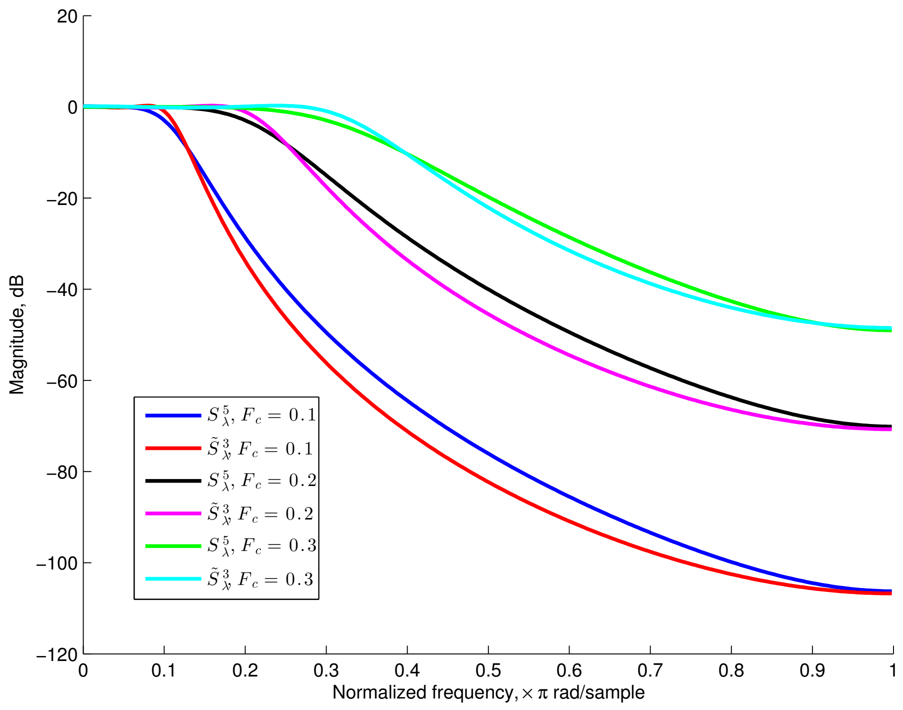


Fig. 4.12: Comparison of the amplitude frequency response of the solution $\tilde{S}_\lambda^3$ of the formulation (4.13) and the standard smoothing spline of order 5 ($p = 3$) evaluated for multiple values of the cut-off frequency

version of Gaussian noise with standard deviation of 0.3. The noise was filtered by a high-pass Butterworth filter with cut-off frequency of 0.2 $\pi rad/sample$ to simulate a situation when the amplitude of the noise increases with frequency. The bandwidth of the signal was about 0.07 $\pi rad/sample$ that fits the frequency range where the smoothing spline filter $\tilde{S}_\lambda^3$ can be controlled. The result of this particular experiment, which is visualized in Fig. 4.15,4.16, showed that the spline $\tilde{S}_\lambda^3$ outperforms its conventional counterpart S_λ^3 in quality of signal reconstruction. $RMSE$ corresponding to the minimum of MSE computed based on the true solution was 3.33% and 5.38% for $\tilde{S}_\lambda^3$ and S_λ^3 respectively. $RMSE$ values for GCV-optimized solutions were 4.6% for $\tilde{S}_\lambda^3$ versus 7.15% for the conventional cubic smoothing spline. Our other experiments showed that, in terms of the reconstruction error, the spline $\tilde{S}_\lambda^3$ performs even better than, the 5-th order spline S_λ^5.

To be able to compute the smoothing spline S_λ^3 with high efficiency we used a B-spline approximation of the dependencies $\lambda_j(F_c)$. In this case GCV and MSE cost functions were optimized in respect to the value of F_c.

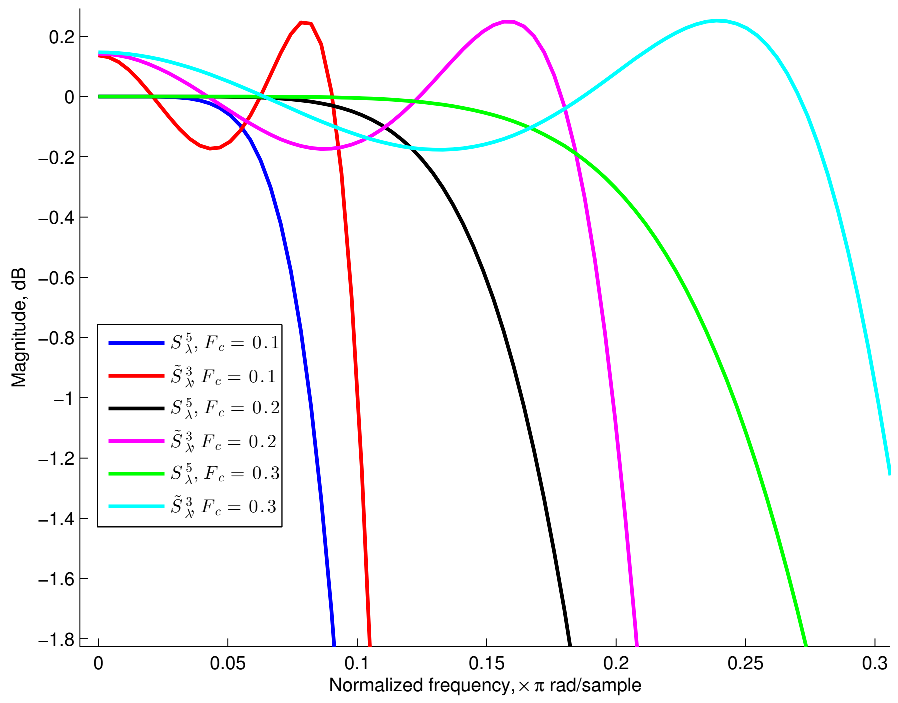


Fig. 4.13: Ripples of the amplitude frequency response in the pass-band for the solution $\tilde{S}_\lambda^3$ of the formulation (4.13)

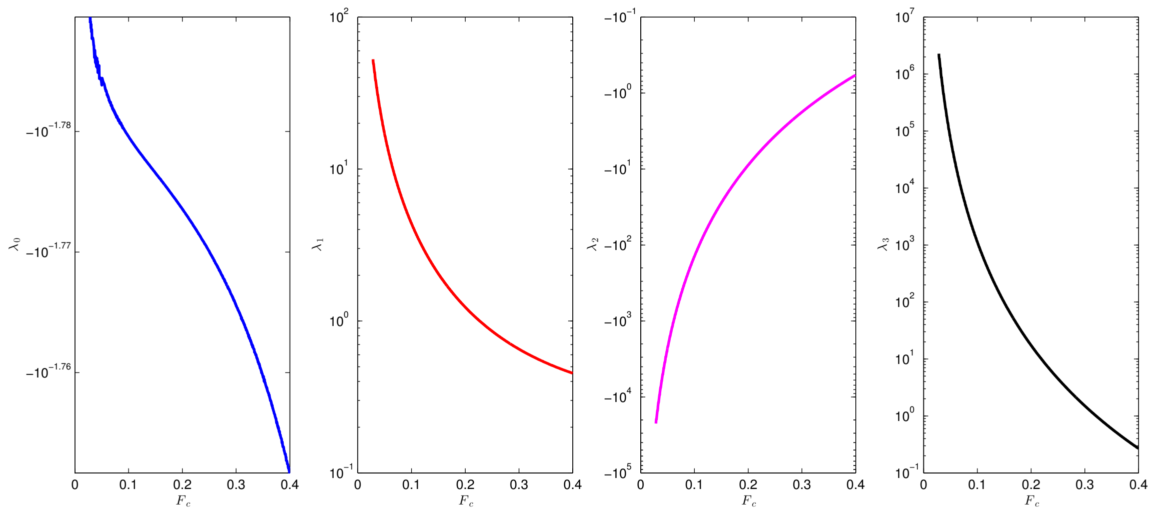


Fig. 4.14: Numerically computed dependencies of regularization parameters λ_j on the value of the cut-off frequency for the extended smoothing spline filter $\tilde{S}_\lambda^3$

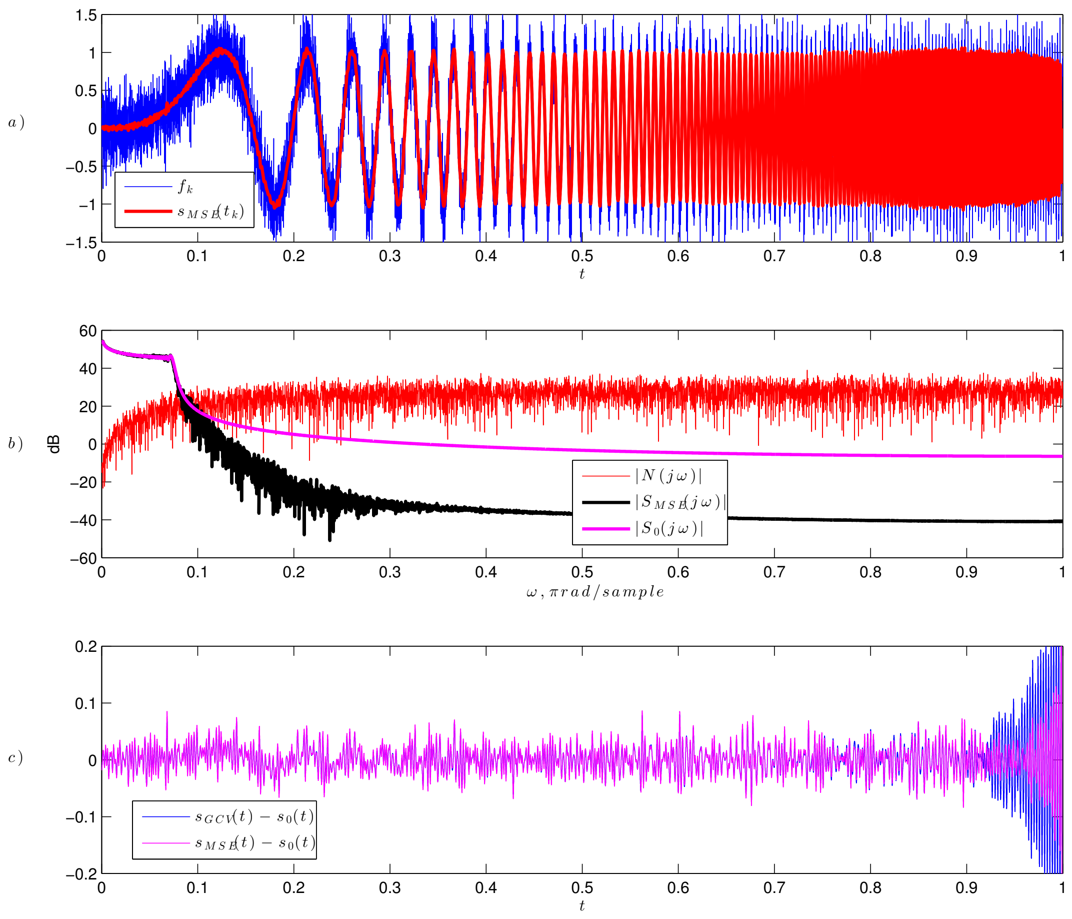


Fig. 4.15: An example of reconstruction of a signal contaminated by Gaussian noise with standard deviation 0.3 using an extended smoothing spline filter $\tilde{S}_\lambda$ a) Noisy signal measurements and the result of the reconstruction corresponding to the minimal value of the MSE cost; b) Amplitude spectrums of the noise, the original signal $s_0(t)$ and the signal contaminated by noise c) reconstruction error computed based on GCV and MSE optimized smoothing spline solutions

4.1.4 Up-sampling by an integer factor

So far we considered the case when the nodes of the sampling grid coincide with the nodes of the target reconstruction grid. Let us now look how the same variational formulation (4.1) can be used for regularized up-sampling of the measurements $\{t_k, f_k\}$ by an integer factor of m. In this case the reconstruction grid step size is m times smaller that the sampling interval: $t_k = t_0 + kmh$. Using the notation presented in [6] we can rewrite the cost function (4.1) as follows:

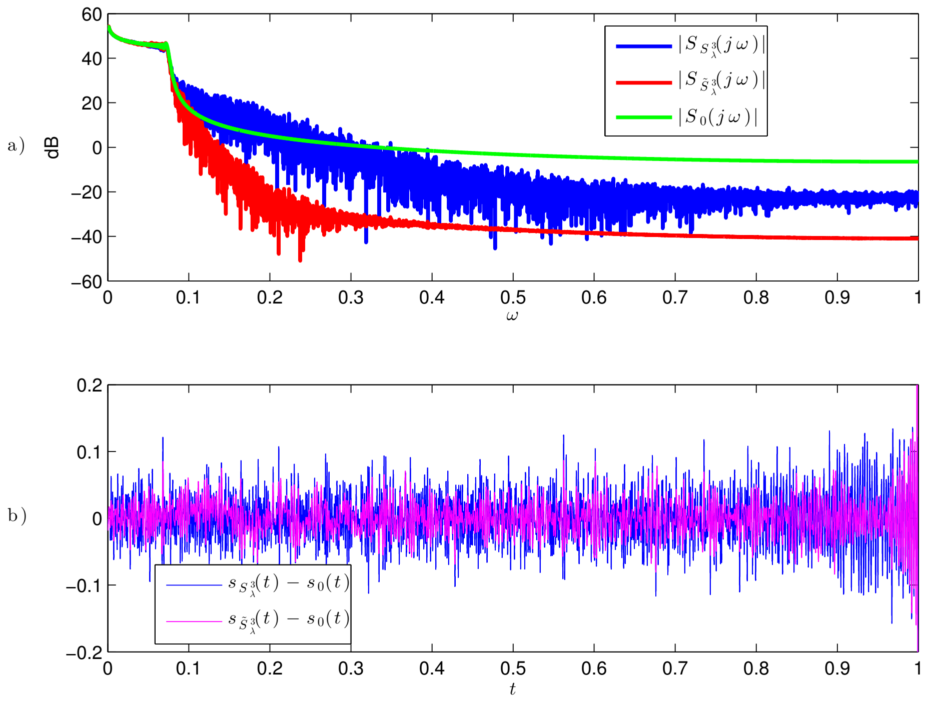


Fig. 4.16: Visualization of the comparison of conventional smoothing spline solution $s_{S^3_\lambda}(t)$ and a spline solution $s_{\tilde{S}^3_\lambda}(t)$ computed based on an extended spline filter $\tilde{S}^3_\lambda$. a) Amplitude spectrums of $s_{S^3_\lambda}(t)$, $s_{\tilde{S}^3_\lambda}(t)$ and the original signal $s_0(t)$ b) The reconstruction error corresponding to the two spline solutions

$$\begin{aligned} J(s) = \langle f(k) - [b_1^n * c(i)]_{\downarrow m}, f(k) - [b_1^n * c(i)]_{\downarrow m}\rangle + \lambda \langle r * c(i), c(i)\rangle = \\ \langle f, f\rangle - 2\langle f, [b_1^n * c]_{\downarrow m}\rangle + \langle [b_1^n * c]_{\downarrow m}, [b_1^n * c]_{\downarrow m}\rangle + \langle r * c, c\rangle \end{aligned} \tag{4.19}$$

Where r is the impulse response of the regularization filter that is not necessarily of the standard form as in (4.1). By differentiating in respect to the unknown B-spline coefficients c and by setting the computed derivative to zero we get the following equation:

$$b_1^n * [[b_1^n * c]_{\downarrow m}]_{\uparrow m} + r * c = b_1^n * [f]_{\uparrow m} \tag{4.20}$$

In z-domain the equation (4.20) can be represented as follows:

$$B_1^n(z)\left(\frac{1}{m}\sum_{k=0}^{m-1} B_1^n(w_k z)C(w_k z)\right) + R(z)C(z) = B_1^n(z)F(z^m) \tag{4.21}$$

where $w_k = e^{-j2\pi k/m}$. The sum $\frac{1}{m}\sum_{k=0}^{m-1} B_1^n(w_k z)C(w_k z)$ in (4.21) corresponds to $[[b_1^n * c]_{\downarrow m}]_{\uparrow m}$. This operation is equivalent to replacing of every m-th sample of the convolution result $b_1^n * c$ by zero. By rearranging the terms and using the properties of z-transform the solution can be expressed as follows:

$$C(z) = \frac{B_1^n(z)}{G_i(z) + R(z)} F(z^m) \tag{4.22}$$

$$G_i(z) = \frac{1}{m} \sum_{k=0}^{m-1} G_{ki}(z) \tag{4.23}$$

In order to understand the structure of the filter with transfer function $G_i(z)$ let us analyze the application of this filter to a signal with z-transform $X(z)$:

$$Y(z) = \frac{1}{m} \sum_{k=0}^{m-1} G_{ki}(z) X(z) = \frac{1}{m} \sum_{k=0}^{m-1} B_1^n(z) B_1^n(w_k z) X(w_k z) = \frac{1}{m} \sum_{k=0}^{m-1} \hat{B}_k(z) \hat{X}_k(z) \tag{4.24}$$

$$\hat{b}_k(l) = \sum_{l_1=-\lfloor n/2 \rfloor}^{\lfloor n/2 \rfloor} b_1^n(l_1) b_1^n(l + l_1) w_k^{-l+l_1} \tag{4.25}$$

$$\hat{X}_k(z) = \sum_{i=-\infty}^{\infty} \hat{x}_k(i) z^i = \sum_{i=-\infty}^{\infty} \left(x(i) w_k^{-i} \right) z^i \tag{4.26}$$

The product $Y_k(z) = \hat{B}_k(z)\hat{X}_k(z)$ can be expressed in time (spatial) domain as:

$$y_k(i) = \sum_{l=-2\lfloor n/2 \rfloor}^{2\lfloor n/2 \rfloor} \hat{b}_k(l) x(i+l) w_k^{-i+l} = \sum_{l=-2\lfloor n/2 \rfloor}^{2\lfloor n/2 \rfloor} \left(\hat{b}(l) w_k^{-i+l} \right) x(i+l) =$$
$$\sum_{l=-2\lfloor n/2 \rfloor}^{2\lfloor n/2 \rfloor} g_{ki}(l) x(i+l) = g_{ki} * x(i) \tag{4.27}$$

Using (4.25) the sequence $g_{ki}(l)$ can be expressed as:

$$g_{ki}(l) = \sum_{l_1=-\lfloor n/2 \rfloor}^{\lfloor n/2 \rfloor} b_1^n(l_1) b_1^n(l + l_1) w_k^{-l+l_1} w_k^{-i+l} = \tag{4.28}$$
$$\sum_{l_1=-\lfloor n/2 \rfloor}^{\lfloor n/2 \rfloor} \left(b_1^n(l_1) w_k^{-i+l_1} \right) b_1^n(l + l_1) = \tilde{b}_{ki} * b_1^n(l)$$

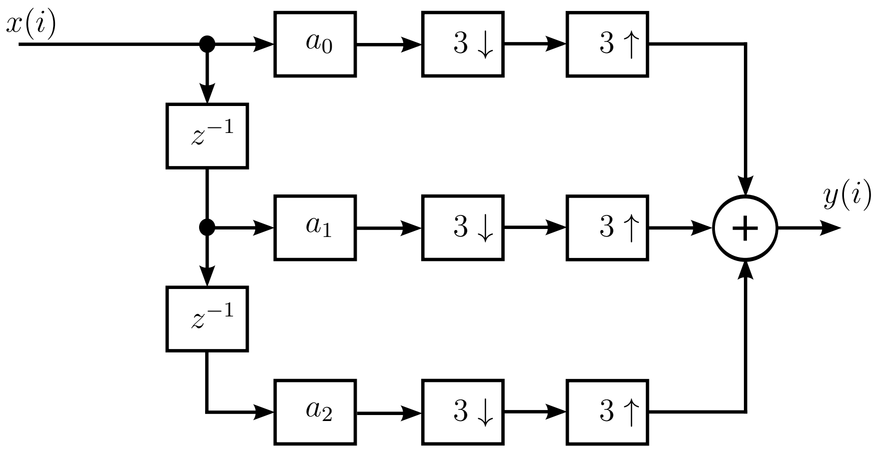
$$\tilde{b}_{ki}(l) = b_1^n(l) w_k^{-i+l} \tag{4.29}$$

After summation over k we get:

$$\tilde{g}_i(l) = \sum_{k=0}^{m-1} g_{ki}(l) = \sum_{k=0}^{m-1} \tilde{b}_{ki} * b_1^n(l) = b_1^n * \left(\left(\sum_{k=0}^{m-1} w_k^{-i+l} \right) b_1^n(l) \right) = b_1^n * \left(v_i(l) b_1^n(l) \right) \tag{4.30}$$
$$i = 0 \ldots m-1, l = -2\lfloor n/2 \rfloor \ldots 2\lfloor n/2 \rfloor$$

Table 4.1: Values of the sequence $v_i(l)$ from (4.30) for cubic spline b_1^3 with various choices of up-sampling factor m

i	$m=2$	$m=3$	$m=4$	$m=5$
0	[0,1,0]	[0,1,0]	[0,1,0]	[0,1,0]
1	[1,0,1]	[0,0,1]	[0,0,1]	[0,0,1]
2		[1,0,0]	[0,0,0]	[0,0,0]
3			[1,0,0]	[0,0,0]
4				[1,0,0]



(a)

Fig. 4.17: Block diagram explaining the structure of the filter with transfer function $A_i(z) = G_i(z) + R(z)$ from equation (4.23) for the case of $m = 3$

The sequence $v_i(l) = \sum_{k=0}^{m-1} w_k^{-i+l}$ is real due to the fact that the filter $\tilde{g}_i(l)$ is real. This means that the sums of imaginary parts of the weights w_k^{-i+l} will vanish. The values of $v_i(l)$ obey a very simple structure that is presented in Table 4.1 for multiple values of m.

From the equation (4.30) it follows that the filter with transfer function $G_i(z)$ has impulse response that changes with every $i-th$ input sample (that is why we use i in designation of $G_i(z)$ and $\tilde{g}_i$). Due to periodicity of the weights $w_k = e^{-j2\pi k/m}$ the impulse response will also change periodically with the period of m. Thus, the linear system $A_i(z) = G_i(z) + R(z)$ in the denominator of (4.23) is represented by m filters $a_i = \tilde{g}_i + r$ and has a polyphase structure. A block-diagram explaining the structure of this filter for $m = 3$ is presented in Fig. 4.17.

We found that it is possible to decompose the filter $A_i(z)$ as a convolution of a causal and a non-causal filter with same polyphase structure. This decomposition can be computed using the following modification of the recursion (4.5):

$$\hat{l}(0)_u = \sqrt{a_{u\,mod\,m}(0) - \sum_{j=1}^{n} \hat{l}(j)_u^2} \tag{4.31}$$

$$\hat{l}(v)_{u+v} = \left(a_{u\,mod\,m}(v) - \sum_{j=1}^{n-m} \hat{l}(j)_i \hat{l}(v+j)_{u+v} \right) \frac{1}{\hat{l}(0)_u}, \tag{4.32}$$

$$v = 1 \dots n, u = 0 \dots$$

After mU iterations of the recursion (4.31) the resulting polyphase decomposition filters are determined as $l_i = \hat{l}_{U(m-1)+i}$. The quality of approximation can be estimated by filtering the Kronecker delta pulse by filters l_i forwards and backwards and by comparing the result with impulse response of the filter a_i. The process of such filtering is described by the following equations:

$$\tilde{c}(i) = \frac{1}{l_{i\,mod\,m}(0)} \left(f(i) - \sum_{v=1}^{n} l_{i\,mod\,m}(v) \right) \tilde{c}(i-v), i = 0 \dots K-1 \tag{4.33}$$

$$c(i) = \frac{1}{l_{i\,mod\,m}(0)} \left(\tilde{c}(i) - \sum_{v=1}^{n} l_{i\,mod\,m}(v) \right) c(i+v), i = K-1 \dots 0 \tag{4.34}$$

4.1.5 Resampling by a rational factor

Now we consider the case when the sampling and reconstruction grid step sizes are related by a rational factor $\frac{h_s}{h} = \frac{m}{q}, m \in \mathbb{N}, q \in \mathbb{N}$. Using the same approach to the analysis of the problem structure as presented above one can show that the smoothing spline filtering operator for this case has the same polyphase structure with m filters $\tilde{g}_i$. We found that these filters can be computed using the following equation:

$$\tilde{g}_i(l) = \sum_{k=-p}^{p} \beta^n \left(\frac{km+i}{q} \right) \beta^n \left(\frac{km+i}{q} + l \right), l = -p \dots p \tag{4.35}$$

The recursion (4.31) and the filtering algorithm described by (4.33,4.34) can be applied exactly in the same way to compute the inverse filter decomposition and to obtain the corresponding spline solution. Since the inverse smoothing spline filter has a polyphase structure the degrees of freedom $df = tr\,(\mathbf{S}_\lambda)$ is proportional to the sum of zero-time lag values of m impulse responses. This facilitates efficient computation of GCV-optimized spline solutions.

4.2 1-D non-uniform smoothing spline filters

In the previous sections we performed a detailed analysis of the uniform smoothing spline formulation. We found that the inverse spline filters can be efficiently computed using

Cholesky-based recursion (4.5). The corresponding spline solutions are determined by a simple and efficient recursive filtering procedure. In a more general setting of resampling by a rational factor $\frac{m}{q}$ we showed that the inverse filters have a polyphase structure and are defined by a set of m filters that can be also efficiently applied using the recursive polyphase filtering algorithm (4.33,4.34).

4.2.1 Solving algorithms

We found that in the most general case of non-uniform sampling when the signal measurements f_k are located in an arbitrary way the situation is not very much different. The spline solution can be found using the following equation in z-domain:

$$C(z) = \frac{1}{G_i(z) + R(z)} \left(\tilde{B}_k(z) F(z) \right) \tag{4.36}$$

Similarly to the previously analyzed cases of signal resampling the filter with transfer function $G_i(z)$ has a polyphase structure, with the only difference that the number of filters in the set is equal to the number of B-spline coefficients N. The filter in the denominator can be decomposed using the same recursion (4.5) with $p = n$, but in contrast to the cases considered above we have to store the impulse responses $\hat{l}_u$ for every iteration number $v = 0 \dots N-1$. The polyphase filter with transfer function $G_i(z)$ can be computed using the following equation:

$$\tilde{g}_i(l) = \sum_k \beta^n(i - t_k)\beta^n(i - t_k + l), \forall k : |t_k - i| \leq n-1, l = -n \dots n \tag{4.37}$$

After factorization of the filter with transfer function $A_i(z) = G_i(z) + R(z)$ using the recursion (4.5) the computation of the spline solution can be performed using the following efficient recursive filtering algorithm:

$$\tilde{c}(i) = \frac{1}{\hat{l}_i(0)} \left(\tilde{f}(i) - \sum_{v=1}^{p} \hat{l}_i(v)\tilde{c}(i-v) \right), i = 0 \dots N-1 \tag{4.38}$$

$$c(i) = \frac{1}{\hat{l}_i(0)} \left(\tilde{c}(i) - \sum_{v=1}^{p} \hat{l}_{i+v}(v)c(i+v) \right), i = N-1 \dots 0 \tag{4.39}$$

As we see from (4.39), the non-causal polyphase filter is different from its causal counterpart. However, due to the overall symmetry of the operator $A_i(z)$ the two filters have a simple relation. The discrete signal $\tilde{f}(i)$ that is passed to the causal filtering part (4.38) of the solving algorithm corresponds to the right hand side term $\tilde{B}_k(z)F(z)$ of equation (4.36). The filter with transfer function $\tilde{B}_k(z)$ also has a polyphase structure. The impulse response of this filter changes with every k-th sample of the non-uniform sequence of measurements f_k. In fact, the operator $\tilde{B}_k(z)$ performs a transformation from the non-uniform sampling grid to the uniform reconstruction grid and therefore, acts as

```
for $k = 0$ to $K-1$ do
    for $l = 0$ to $n$ do
        $\tilde{f}(e_k + l) \leftarrow \tilde{f}(e_k + l) + f_k \tilde{b}_k(l)$
    end for
end for
```

Fig. 4.18: An algorithm for computing discrete sequence $\tilde{f}_k$ corresponding to the right hand side $\tilde{B}_k(z)F(z)$ of the equation (4.36)

a variable rate conversion filter. This operation can be performed efficiently using an algorithm presented in Fig. 4.18, where a set of FIR filters $\tilde{b}_k$ and a discrete sequence e_k are defined as follows:

$$e_k = \begin{cases} \lfloor \frac{t_k}{h} + 0.5 \rfloor - \frac{n}{2} & \text{if } n \text{ is even} \\ \lfloor \frac{t_k}{h} \rfloor - \frac{n-1}{2} & \text{if } n \text{ is odd} \end{cases} \tag{4.40}$$

$$\tilde{b}_k(l) = \beta^n \left(\frac{t_k}{h} - e_k - l \right), l = 0 \dots n \tag{4.41}$$

For the sake of example in Listing 4.1 we provide an implementation of this algorithm in the MathOberon language [63]. The proposed right hand side computation algorithm is based on scaling and increment operations applied to vectors of size $n+1$. This can be done efficiently using the "AXPY" BLAS level-1 routine [64]. Moreover, in the case of cubic B-splines the size of processed vectors is 4, which is the typical size of Single Instruction Multiple Data (SIMD) engines in current computing systems such as multi-core CPU and GPGPU. This, in turn, allows to implement the right hand side computation algorithm with high efficiency.

```
procedure ComputeRhs(
                                const f: array [K] of real; (* samples f(k) *)
                                const tbk: array [K,n+1] of real; (* filters b̃k(l) *)
                                const ek: array [K] of integer; (* sequence ek(i) *)
                                var tf: array [N] of real (* right hand side f̃(i) *)
                                )
var i, k: integer;
begin
    tf [..] := 0;
    for k := 0 to K−1 do
        i := ek[k];
        tf[i .. i+n] := tf[i .. i+n] + f[k] * tbk[k ,..];
    end;
end ComputeRhs;
```

Listing 4.1: A MathOberon implementation of the algorithm for computing discrete sequence $\tilde{f}_k$ corresponding to the right hand side $\tilde{B}_k(z)F(z)$ of the equation (4.36)

Depending on an application one can use various strategies for optimizing the computation and the handling of the presented non-uniform smoothing splines operators. When a spline operator is repeatedly used to compute multiple solutions, it makes sense to precompute the filters $\hat{l}_i$ and store them in memory for future reuse. This requires to store $(n+1)N$ floating point values. In this particular case the filters $a_i = \tilde{g}_i + r$

can be computed online based on the equations (4.37,4.14,4.15) without explicit storage. However, when the regularization term is manipulated by changing the regularization parameter(s), it is more beneficial to store the filters a_i with the same memory requirement of $(n+1)N$ elements.

Note, that the presented solving algorithms are formally equivalent to the process of solving a linear system of equations with a sparse symmetric positive definite matrix corresponding to the operator $A(z)$ using Cholesky decomposition. However, practically our algorithms do not require handling of any sparse matrices and due to the explicit use of the specific problem structure allow to achieve higher performance than highly optimized sparse matrix packages.

4.2.2 Square-root free decomposition algorithm

On some computing platforms the complexity of the square root operation in the recursion (4.5) may limit maximal achievable performance. This is the case of embedded systems where the square root is either emulated using software or requires many processor cycles for its evaluation. In such cases the recursion (4.5) can be modified in a way that avoids the use of the square root. This feature comes at the cost of an additional $\frac{n^2+n}{2}$ number of multiplications. The modified recursion is described by the following equations:

$$\hat{l}(0)_u = a(0) - \sum_{j=1}^{q} \hat{l}(0)_{u-j}\hat{l}(j)_u^2 \tag{4.42}$$

$$\hat{l}(v)_{u+v} = \left(a(v) - \sum_{j=1}^{q-v} \hat{l}(0)_{u-j}\hat{l}(j)_u\hat{l}(v+j)_{u+m} \right) \frac{1}{\hat{l}(0)_u}, v = 1\dots q, u = 0\dots$$

where $q = p$ for the case of conventional smoothing splines and $q = n$ corresponds to the cases of resampling and non-uniform smoothing splines. The filtering algorithm is modified accordingly. For non-uniform smoothing splines it can be represented as follows:

$$\tilde{c}(i) = \tilde{f}(i) - \sum_{v=1}^{n} \hat{l}_i(v)\tilde{c}(i-v), i = 0\dots N-1 \tag{4.43}$$

$$c(i) = \frac{1}{\hat{l}_i(0)}\tilde{c}(i) - \sum_{v=1}^{n} \hat{l}_{i+v}(v)c(i+v), i = N-1\dots 0 \tag{4.44}$$

4.2.3 Software implementation and performance evaluation

We implemented the proposed algorithms for non-uniform spline reconstruction in Math-Oberon. We evaluated the computational performance of our implementation on a PC platform equipped with an Intel i7-3770 3.4 GHz Quad-Core processor and dual channel DDR3 800 MHz memory. The results of the evaluation for the solving process that includes the computation of the right hand side and inverse filtering are presented in Fig. 4.19 and Fig. 4.20. These processes were optimized with the use of SIMD operations

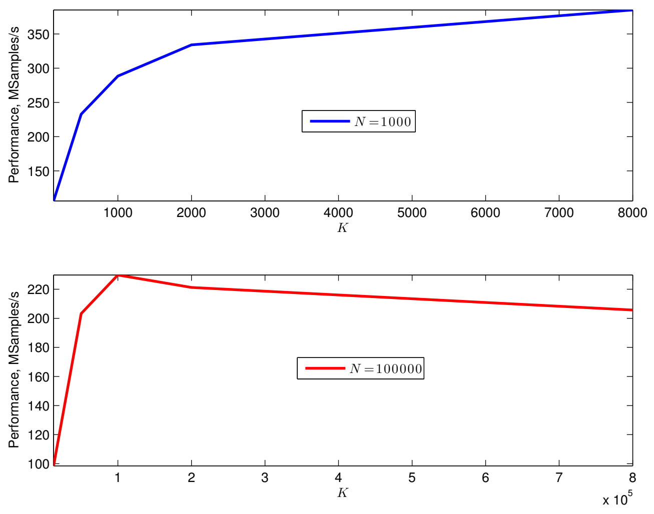


Fig. 4.19: Evaluated performance in MSamples/s for our MathOberon implementation of the non-uniform spline solving operator for multiple values of number of samples K and two choices of the reconstruction grid size: $N = 1000$ and $N = 100000$

supported by the processor. Our non-optimized Cholesky-based inverse filter decomposition algorithm implementation is about an order of magnitude faster in terms of execution time than Matlab's `chol` function implemented based on CHOLMOD – one of the best available sparse Cholesky algorithm implementations [65, 66]. The high-performance achieved with our algorithms allows efficient application of non-uniform smoothing splines in practical signal processing applications.

4.3 Application to medical Optical Coherence Tomography (OCT)

Optical Coherence Tomography (OCT) is a modern technology that allows to perform high-resolution, cross-sectional tomographic images of the internal microstructure in materials and biologic systems by measuring backscattered or backreflected light. Its technical advantages over other existing imaging modalities in combination with non-invasiveness allowed a successful application of this technology in the fields of medicine and biomedical research [67, 68, 69].

Fourier domain OCT (FD-OCT) is one of the most popular and efficient techniques used for implementing modern medical OCT systems. In this technique the signal acquired at a regular wavelength grid is reconstructed using the Fast Fourier Transform (FFT). The application of FFT requires to transform (remap) the wavelength-equidistant measurements to a uniform grid in wavenumbers $k = \frac{2\pi}{\lambda}$, so called k-space. The quality of this transformation has a significant influence on the quality of overall tomographic

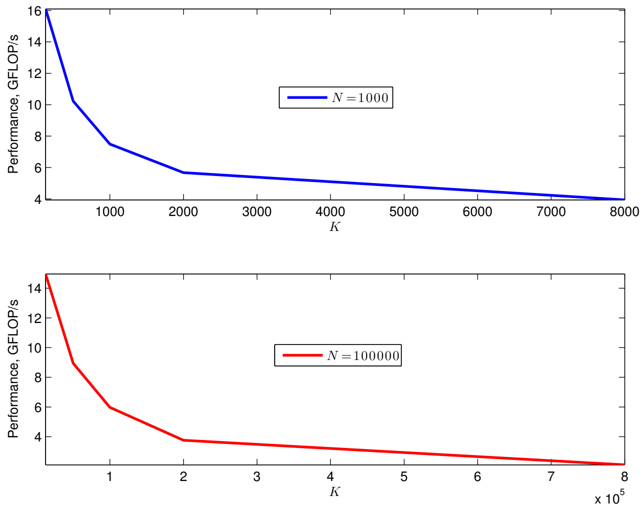


Fig. 4.20: Evaluated performance in GFLOP/s for our MathOberon implementation of the non-uniform spline solving operator for multiple values of number of samples K and two choices of the reconstruction grid size: $N = 1000$ and $N = 100000$. For computing the numbers in GFLOP we used the fact that the right hand side computation and inverse filtering algorithms require $2K(n+1)$ and $2N(2n+1)$ floating point operations respectively

reconstruction. The standard existing approaches to this problem include spline-based interpolation and non-uniform fast Fourier transform (NUFFT) [70, 71, 72]. Besides the quality of grid transformation, another key requirement is the ability to perform this process with high performance. This is critical for real-time reconstruction of OCT images implemented in industrial OCT scanners.

We found that our approach for non-uniform spline-based signal reconstruction can be efficiently applied to solve the described problem. Based on the theory of FD-OCT [73] we created a mathematical model of an FD-OCT imaging system depicted in Fig. 4.21. The model is able to generate synthetic images based on a provided reflectivity function. In this way we facilitated the evaluation of the performance of our algorithm in comparison with other existing techniques.

The evaluation was performed on two synthetic datasets: a dataset with a diagonal line pattern and a dataset generated based on a real high-quality OCT image of retina. For both datasets the generated scan-lines had 2048 equidistant samples within the wavelength range of $[0.850, 0.950]\mu m$. Before application of the Fourier-based reconstruction method the generated data was processed by three grid transformation algorithms: cubic spline interpolation, one of the best existing NUFFT algorithms [71] and our non-uniform spline approximation algorithm introduced in Section 4.2.

From the evaluation results that are visualized in Fig. 4.22,4.23,4.24,4.25 one can see that our algorithm offers higher quality of reconstruction in comparison with the conven-

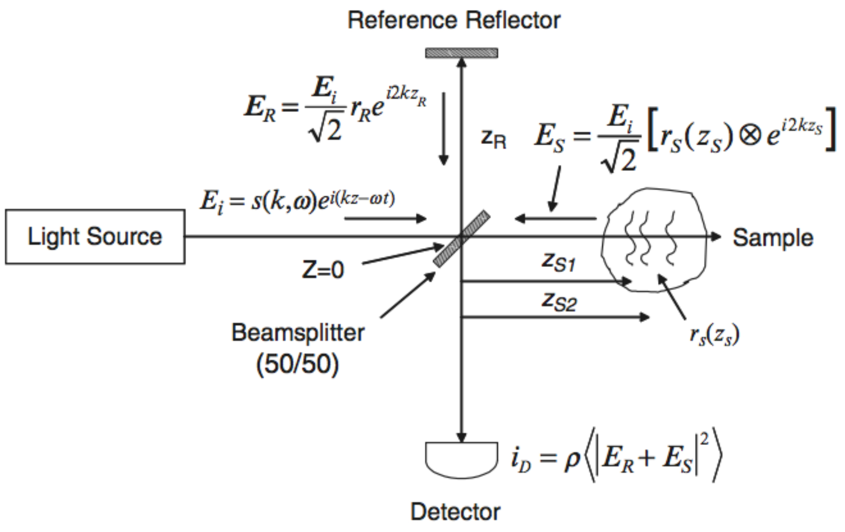

Reference Reflector
$E_R = \frac{E_i}{\sqrt{2}} r_R e^{i2kz_R}$
z_R
$E_S = \frac{E_i}{\sqrt{2}}[r_S(z_S) \otimes e^{i2kz_S}]$
$E_i = s(k,\omega)e^{i(kz-\omega t)}$
Light Source
Sample
Z=0
z_{S1}
z_{S2}
Beamsplitter (50/50)
$r_s(z_s)$
$i_D = \rho\langle|E_R + E_S|^2\rangle$
Detector

Fig. 4.21: A typical frequency domain OCT imaging setup based on Michelson interferometer

tional spline interpolation and NUFFT. Moreover, our solving algorithm implemented in Matlab was about 6 times faster than Matlab's `interp1` function used for cubic spline interpolation and about 8 times faster than the used NUFFT algorithm.

4.4 Uniform smoothing spline filters in higher dimensions

In higher dimensions the cost function of the one-dimensional smoothing spline (4.1) can be represented in the following general form:

$$J(s) = \sum_{\mathbf{k}} |s(\boldsymbol{t}_{\mathbf{k}}) - f_{\mathbf{k}}|^2 + \int_{\mathbb{R}} \|D_{\boldsymbol{\lambda}} s(\boldsymbol{t})\|^2 \mathrm{d}\boldsymbol{t} \tag{4.45}$$

where $\boldsymbol{t}_{\mathbf{k}} \in R^d \times \mathbb{R}^{K_1} \times \ldots \times \mathbb{R}^{K_d}$, $f_{\mathbf{k}} \in \mathbb{R}^{K_1} \times \ldots \times \mathbb{R}^{K_d}$, $D_{\boldsymbol{\lambda}}$ is some differential operator, as for example, a Duchon seminorm of order p:

$$\int_{\mathbb{R}} \|D_{\boldsymbol{\lambda}} s(\boldsymbol{t})\|^2 \mathrm{d}t = \lambda \sum_{p_1+\cdots+p_d=p} \frac{p!}{p_1! \cdots p_d!} \int_{\mathbb{R}^d} \left(\frac{\partial^p s(\boldsymbol{t})}{\partial t_1^{p_1} \cdots \partial t_d^{p_d}} \right)^2 \mathrm{d}\boldsymbol{t}. \tag{4.46}$$

or a modified version of the Laplacian regularization operator:

$$\int_{\mathbb{R}} \|D_{\boldsymbol{\lambda}} s(\boldsymbol{t})\|^2 dt = \sum_{m=1}^{d} \lambda_m \int_{\mathbb{R}^d} \left(\frac{\partial^2 s(\boldsymbol{t})}{\partial^2 t_m} \right)^2 \mathrm{d}\boldsymbol{t} \tag{4.47}$$

The unknown signal $s(\boldsymbol{t}), \boldsymbol{t} \in \mathbb{R}^d$ is represented using a tensor product of B-splines:

$$s(\boldsymbol{t}) = \sum_{i_1 \in \mathbb{Z}, i_2 \in \mathbb{Z}} c(i_1, i_2) \beta^{n_1}(t_1/h_1 - i_1) \beta^{n_2}(t_2/h_2 - i_2) \tag{4.48}$$

Usually the degrees of B-spline functions in the product are chosen to be equal: $n_1 = n_2$.

Unser et all. showed in [6] that for some particular cases of the differential operator $D_{\boldsymbol{\lambda}}$ the uniform smoothing spline solution can be determined by successive application of the one-dimensional inverse smoothing spline operator along the dimensions of the array of signal measurements. For example, in 2-D case this corresponds to the following form of the regularization term:

$$\int_{\mathbb{R}} \|D_{\boldsymbol{\lambda}} s(\boldsymbol{t})\|^2 dt = \lambda_1 \int_{\mathbb{R}^2} \left(\frac{\partial^p s(\boldsymbol{t})}{\partial t_1^p} \right)^2 \mathrm{d}\boldsymbol{t} + \lambda_2 \int_{\mathbb{R}^2} \left(\frac{\partial^p s(\boldsymbol{t})}{\partial t_2^p} \right)^2 \mathrm{d}\boldsymbol{t} + \lambda_1 \lambda_2 \int_{\mathbb{R}^2} \left(\frac{\partial^p s(\boldsymbol{t})}{\partial t_1^{p/2} \partial t_2^{p/2}} \right)^2 \mathrm{d}\boldsymbol{t} \tag{4.49}$$

where λ_1 and λ_2 are regularization parameters for first and second dimension respectively. Using one-dimensional inverse filtering algorithms presented above, such separable smoothing splines can be computed with high efficiency. Moreover, due to separability the effective degrees of freedom $df = tr\,(\mathbf{S}_\lambda)$ is determined by the product of degrees of freedom for one-dimensional smoothing spline operators corresponding to each data dimension. This facilitates fast computation of GCV-optimized approximating spline solutions. Fig. 4.26 presents an example of a 3-D cubic ($p = 2$) GCV-optimized smoothing spline fitted to a medical CT dataset contaminated by Gaussian noise with standard deviation of 30 ($SNR \approx 7.7dB$). In this example the smoothing parameter λ was the same for all dimensions. GCV-optimized value of λ was ~ 1.0, the corresponding $RMSE$ was ~ 7.53 % ($SNR \approx 22.46$ dB).

4.4.1 2-D case

Let us consider the case of computing 2-D smoothing spline of degree n:

$$C(z_1, z_2) = \frac{B^{n_1,n_2}(z_1, z_2)}{B^{n_1,n_2}(z_1, z_2) B^{n_1,n_2}(z_1, z_2) + R_{\boldsymbol{\lambda}}(z_1, z_2)} F(z_1, z_2) \tag{4.50}$$

where $B^{n_1,n_2}(z_1, z_2) = B^{n_1}(z_1) B^{n_2}(z_2)$, $R_{\boldsymbol{\lambda}}(z_1, z_2)$ is a non-separable regularization filter corresponding to the functional $\int_{\mathbb{R}} \|D_{\boldsymbol{\lambda}} s(\boldsymbol{t})\|^2 d\boldsymbol{t}$. The difficulty in this case is that the transfer function of the filter in the denominator $A(z_1, z_2) = B^{n_1,n_2}(z_1, z_2) B^{n_1,n_2}(z_1, z_2) + R_{\boldsymbol{\lambda}}(z_1, z_2)$ in its general form cannot be decomposed to a product of first-order terms as in the case of one dimension. Therefore, the inverse spline filter is not separable. It turned out there exists an alternative non-separable but very efficient way to compute the inverse smoothing spline filter. Let us consider the following form of a decomposition of the filter $A(z_1, z_2)$:

$$A(z_1, z_2) = L(z_1, z_2) L(z_1^{-1}, z_2^{-1}) \tag{4.51}$$

$$L(z_1, z_2) = L_0(z_1) + L_1(z_1) z_2^{-1} + \ldots + L_{n_2}(z_1) z_2^{-n_2} \tag{4.52}$$

where the one-dimensional transfer functions $L_j(z_1)$ have the following form:

$$L_j(z_1) = l_j(0) + l_j(1) z_1^{-1} + \ldots + l_j(n_1) z_1^{-n_1}, j = 0 \ldots n_2 \tag{4.53}$$

If we take a closer look at the equations (4.51,4.52) we can recognize the minimum-phase decomposition from (4.4). The only difference is that instead of scalar coefficients we now have one-dimensional filters with transfer functions $L_j(z_1)$. To compute this decomposition we can use a modified version of Cholesky recursion that in z-domain can be represented as follows:

$$\hat{L}_{0,(i)}(z_1) = Fact_1\left(A_0(z_1) - \sum_{j=1}^{n_2} \hat{L}_{j,(i)}(z_1)\hat{L}_{j,(i)}(z_1^{-1})\right) \tag{4.54}$$

$$\hat{L}_{v,i+v}(z_1) = \left(A_v(z_1) - \sum_{j=1}^{n_2-v} \hat{L}_{j,(i)}(z_1)\hat{L}_{v+j,(i+v)}(z_1^{-1})\right)\frac{1}{\hat{L}_{0,(i)}(z_1)}, \quad v = 1\dots n_2$$

where $Fact_1\,()$ is an operator that performs factorization of a symmetric one-dimensional FIR filter, transfer functions $A_v(z_1)$ are defined by the following equations:

$$A(z_1,z_2) = A_{n_2}(z_1)z_2^{-n_2} + \dots + A_0(z_1) + \dots + A_{n_2}(z_1)z_2^{n_2} \tag{4.55}$$

$$A_v(z_1) = a_v(n_1)z_1^{-n_1} + \dots + a_v(0) + \dots + a_v(n_1)z_1^{n_1}, v = 0\dots n_2 \tag{4.56}$$

In the same way as for (4.5) the sequence $\hat{L}_{v,(i)}(z_1)$ converges to $L_v(z_1)$ with a specified accuracy in a finite number of iterations, which was verified experimentally.

When the decomposition (4.51) is determined, the corresponding smoothing spline solution can be efficiently computed based on the following recursive filtering algorithm:

$$\tilde{c}(i_1,i_2) = \boldsymbol{l}_0^{-1}(i_1) * \left(f(i_1,i_2) - \sum_{v=1}^{n_2} \boldsymbol{l}_v(i_1) * \tilde{c}(i_1,i_2-v)\right), i_2 = 0\dots K_2-1 \tag{4.57}$$

$$c(i_1,i_2) = \boldsymbol{l}_0^{\prime -1}(i_1) * \left(\tilde{c}(i_1,i_2) - \sum_{v=1}^{n_2} \boldsymbol{l}_v^{\prime}(i_1) * c(i_1,i_2+v)\right), i_2 = K_2-1\dots 0 \tag{4.58}$$

where $\boldsymbol{l}_v$ are filters corresponding to $L_v(z_1)$, $'$ symbol is used to denote the adjoint of a discrete signal that is equivalent to signal reversal [6]. As we can see, the presented algorithm is based on two parallel recursive filtering processes: one is based on column-wise (over i_1) filtering using a set of one-dimensional filters $\boldsymbol{l}_v$, while the other recurs along the rows (over i_2) of the filtered signal. Overall, the convolutional structure of both decomposition and inverse filtering algorithms allows to implement non-separable 2-D smothing splines with high-efficiency. If implemented using convolution in time (spatial) domain the filtering algorithm requires the computational complexity of about $4\sum_{v=0}^{n_2} q_v N_1 N_2$ operations. Where q_m is the number of samples for every filter $\boldsymbol{l}_v$ in the inverse spline operator decomposition. In general q_m are infinite due to the fact that filters $\boldsymbol{l}_v$ are IIR filters, which is clearly seen from the recursion (4.54). In practice the effective length of these filters can be relatively small – from few samples to hundreds of samples. This depends on the effective length (radius) of the impulse response corresponding to the regularization operator $R_{\boldsymbol{\lambda}}(z_1,z_2)$ controlled by smoothing parameters $\boldsymbol{\lambda}$: the bigger the values of $\boldsymbol{\lambda}$, the bigger the values of q_m. With relatively large q_v one-dimensional convolutions in (4.54,4.57,4.58) can be computed with higher efficiency using one-dimensional fast Fourier transform (FFT). In this case the complexity of the inverse filtering process will be about $2(n_2+1)\left(2Clog_2(N_1)+1\right)N_1N_2$, where for FFTW software library [74] C is 2.5. In case if the inverse smoothing spline filter does not have large effective length it is possible to compute the solution using its 2-D Fourier image. In this case an FFT-based computation will require about $2\left(Clog_2(N_1)N_2+Clog_2(N_2)N_1\right)+N_1N_2$ operations. Memory requirements for storing a non-separable smoothing spline operator for filtering in time (spatial) domain is $\sum_{v=0}^{n_2} q_v$ elements, for one-dimensional FFT-based filtering is $(n_2+1)N_1$ elements and for two-dimensional FFT-based filtering is N_1N_2 elements.

Note, that with the presented approach we get access to a rich variety of approximating spline solutions, which can be constructed based on various differential forms appearing in PDE equations from physical and mathematical problems. Separable multidimensional filters do not offer this. As an example, Fig. 4.27 shows amplitude frequency responses of two 2-D cubic smoothing spline operators: one corresponds to a separable solution, while the other is based on the Duchon seminorm of order $p=2$ and was computed via the recursion (4.54) and by filtering the Kronecker delta pulse using the (4.57), (4.58). From the figure we see that the two splines are very different in terms of the characteristics of their corresponding inverse filters. In contrast to its separable counterpart the Duchon-based spline has a high level of isotropy of attenuation in the frequency domain. Usually, a new quality comes at an expense of decreasing the level of other qualities. It is clear that even with its efficient convolutional structure the filtering algorithm (4.57,4.58) cannot compete with the separable approach in terms of performance.

An interesting property of both decomposition (4.54) and filtering algorithm (4.57, 4.58) is that they can be performed in two different ways. Instead of searching a decomposition of $A(z_1,z_2)$ using (4.52), we could consider another form of the filter $L(z_1,z_2)$:

$$L(z_1,z_2)=L_0(z_2)+L_1(z_2)z_1^{-1}+\ldots+L_{n_1}(z_2)z_1^{-n_1} \tag{4.59}$$

Basically we can switch the direction for performing the decomposition and inverse filtering without changing the resulting spline solution. What does it imply? Fig. 4.28 presents an example of a non-separable smoothing spline operator that is constructed based on the tensor product of B-splines with different degrees $n_1=2, n_2=3$. In this case the tensor filter with transfer function $A(z_1,z_2)$ looses it symmetry. This means that if we use the decomposition with factors of the form (4.52) we will get $n_2+1=4$ filters $\boldsymbol{l}_v$, in case if we have one-dimensional filtering along the second dimension we will get $n_1+1=3$ filters $\tilde{\boldsymbol{l}}_v$. The impulse responses of the filters corresponding to these two cases are shown in Fig. 4.28. It turns out that filters $\boldsymbol{l}_v$ and $\tilde{\boldsymbol{l}}_v$ have approximately the same effective length. This makes decomposition and filtering with one-dimensional computations along the second dimension more efficient. Another example when the direction of inverse filtering can matter assumes the use of FFT. As we explained above the complexity of FFT-based filtering is $2(n_2+1)\left(2Clog_2(N_1)+1\right)N_1N_2$. Now assume that $\left(2Clog_2(N_1)+1\right)N_1 >> \left(2Clog_2(N_2)+1\right)N_2$. In this case it is clear that filtering along the second dimension requires less computations and is therefore more advantageous.

4.4.2 The case of higher dimensions

After we understood the mechanics of the presented non-separable smoothing spline approach in two dimensions, its extension to higher dimensions is straightforward. For the case of an arbitrary number of dimensions d we can use the following recursion:

$$\hat{L}^k_{0,(i)}(\boldsymbol{z}_k) = Fact_k\left(A^k_0(\boldsymbol{z}_k) - \sum_{j=1}^{n_k} \hat{L}^k_{j,(i)}(\boldsymbol{z}_k)\hat{L}^k_{j,(i)}(\boldsymbol{z}_k^{-1})\right) \tag{4.60}$$

$$\hat{L}^k_{v,i+v}(\boldsymbol{z}_k) = \left(A^k_v(\boldsymbol{z}_k) - \sum_{j=1}^{n_k-v} \hat{L}_{j,(i)}(\boldsymbol{z}_k)\hat{L}_{v+j,(i+v)}(\boldsymbol{z}_k^{-1})\right)\frac{1}{\hat{L}_{0,(i)}(\boldsymbol{z}_k)}, \quad v = 1\dots n_k$$

where $A(\boldsymbol{z}_k) \equiv A(z_1,\dots,z_k)$, $Fact_k\,()$ is a factorization operator applied to a k-dimensional symmetric tensor filter. The recursion starts at $k = d-1$ and is propagated to lower dimensions until it reaches $k = 1$. In a similar way we can rewrite the inverse filtering algorithm:

$$\tilde{c}^k(\boldsymbol{i}_k) = \left(\boldsymbol{l}^k_0(\boldsymbol{i}_{k-1})\right)^{-1} * \left(f^k(\boldsymbol{i}_{k-1}) - \sum_{m=1}^{n_k} \boldsymbol{l}^k_m(\boldsymbol{i}_{k-1}) * \tilde{c}^k(\boldsymbol{i}_{k-1}, i_k - m)\right), \quad i_k = 0\dots K_k - 1 \tag{4.61}$$

$$c^k(\boldsymbol{i}_k) = \left(\boldsymbol{l}'^k_0(\boldsymbol{i}_{k-1})\right)^{-1} * \left(\tilde{c}^k(\boldsymbol{i}_k) - \sum_{m=1}^{n_k} (\boldsymbol{l}^k_m)'(\boldsymbol{i}_{k-1}) * c^{k+}(\boldsymbol{i}_{k-1}, i_k + m)\right), \quad i_k = K_k - 1\dots 0 \tag{4.62}$$

where $c^k(\boldsymbol{i}_k) \equiv c^k(i_1,\dots,i_k)$ is a k-dimensional slice of the solution at the k-th level of the recursion, $\boldsymbol{l}^k_m(\boldsymbol{i}_{k-1})$ are $(k-1)$-dimensional filters. At $k = 2$ filtering will be performed according to 2-D algorithm (4.57,4.58).

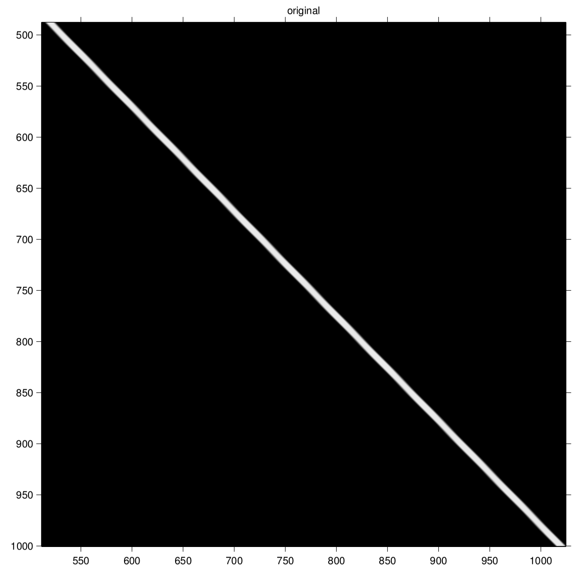
To simplify the estimation of computational and memory requirements of the inverse multidimensional filtering algorithm, let us assume that $n_1 = n_2 = \dots = n_d = n$ and that at every $d-1$ level of the filtering recursion filters $\boldsymbol{l}^k_m(\boldsymbol{i}_{k-1})$ have q number of elements. Therefore, the overall number of one-dimensional filters in the hierarchical decomposition of the filter $A(\boldsymbol{z}_d)$ will be equal to $(n+1)^{d-1}q^{d-2}$ each having q coefficients. Therefore, when computed based on time (spatial) domain 1-D filtering the algorithm will require about $4\left((n+1)q\right)^{d-1}\prod_{k=1}^{d} K_k$. The total number of coefficients to store is $\left((n+1)q\right)^{d-1}$. Similarly to the described case of 2-D filtering, the use of FFT can potentially decrease the computational complexity at the cost of increased memory requirements.

4.5 Non-uniform smoothing spline filters in higher dimensions

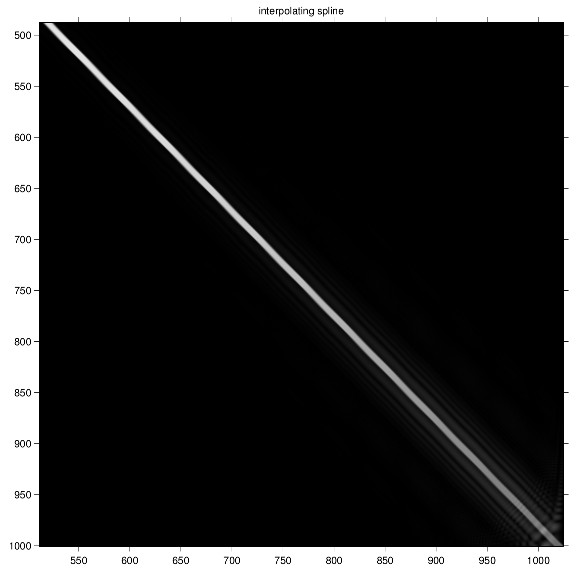
The proposed recursive filtering approach can be extended to the most general case of non-uniform spline reconstruction in multiple dimensions. In this case the decomposition and solving algorithms will have exactly same structure as presented in equations (4.60,4.61,4.62). One major difference is that the one-dimensional filters at the lowest level of the hierarchical structure of the inverse filter will have the form of non-uniform 1-D inverse filters. This implies a significant increase in computational and memory requirements for determining the decomposition and the corresponding spline solution. Within our current research we are investigating strategies for implementing such reconstruction algorithms as efficient as possible.

4.6 Discussion

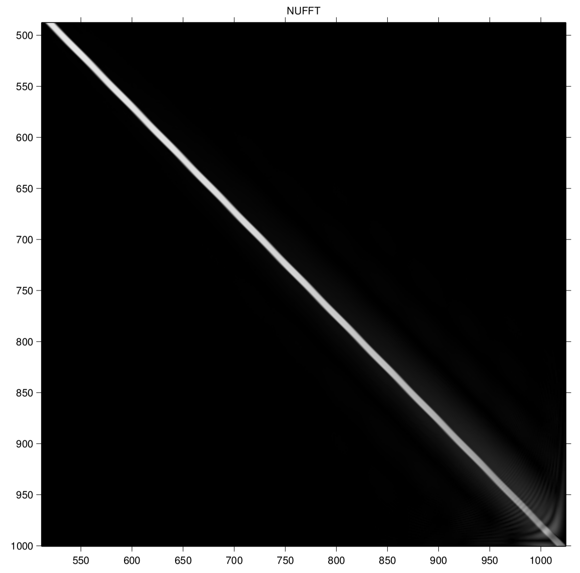
We revisited the problem of spline-based variational signal reconstruction using the approach of inverse recursive filtering. As a result we derived highly efficient recursive filtering algorithms for computing approximating spline solutions of an arbitrary order that are statistically optimal in the sense of minimization of a cross validation-based cost function. We introduced a new family of approximating spline filters with improved noise discrimination characteristics. These filters are determined based on an extension of the classical smoothing spline regularization approach. We extended the results obtained for uniform approximating splines to the case of non-uniform spline reconstruction. This extension led to the derivation of highly efficient algorithms for computing inverse spline filtering operators and the corresponding spline solutions. Finally, we extended the proposed approach to the case of multidimensional signal reconstruction. The achieved results are not limited to only B-spline based signal representation but can be used for any shift-invariant bases. The developed solving algorithms were successfully applied to a medical imaging problem of Optical Coherence Tomography (OCT) and showed significant outperformance compared with conventional techniques used in the field.



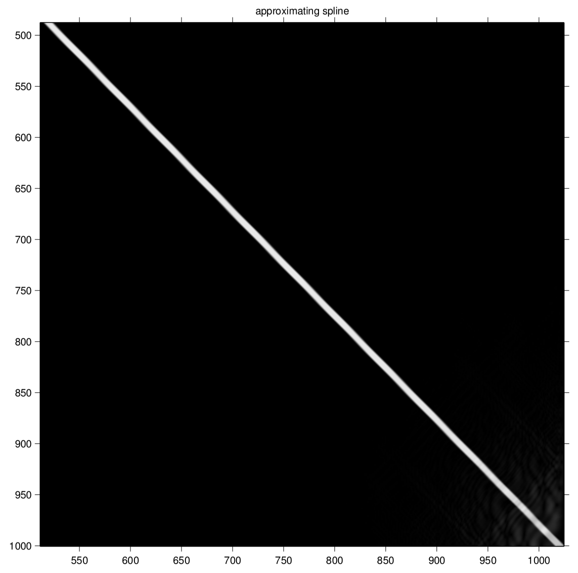
(a)



(b)

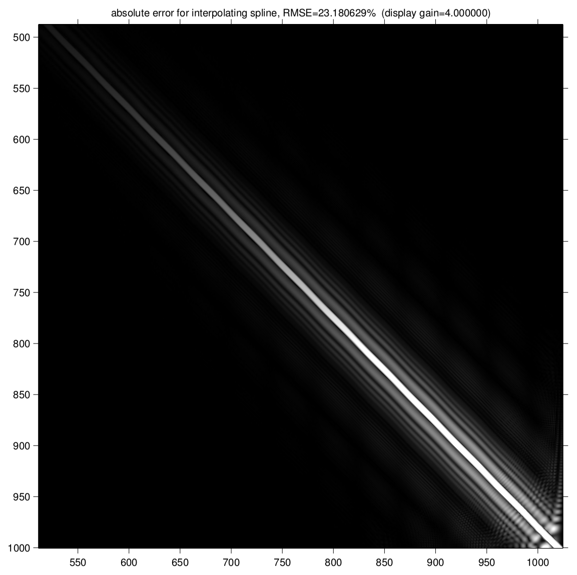


(c)

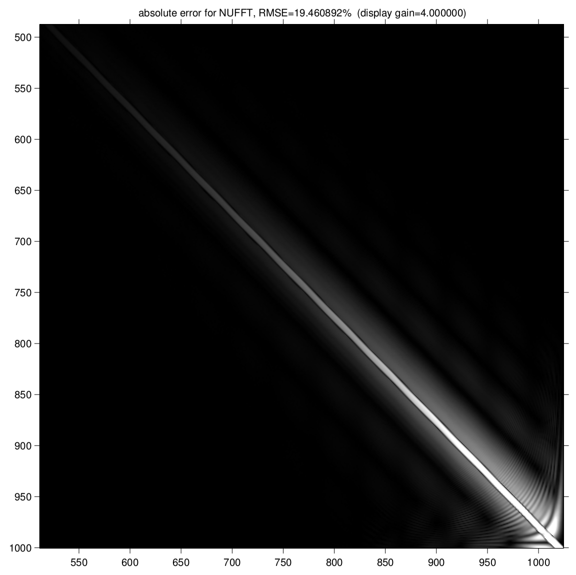


(d)

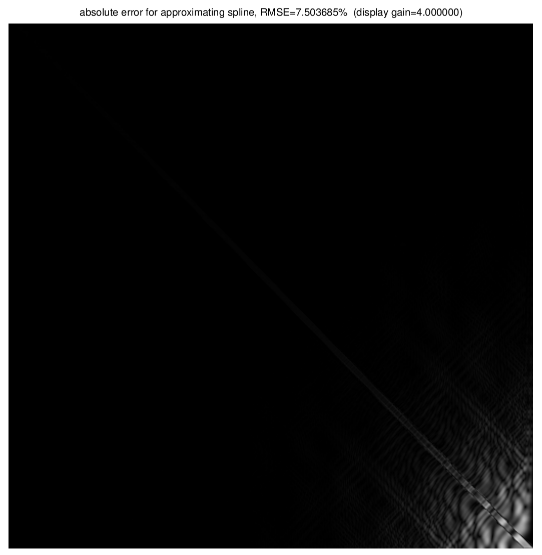
Fig. 4.22: Visualization of the reconstruction of a synthetic OCT data with a diagonal line pattern. (a) the true solution, (b) interpolating spline-based reconstruction, (c) NUFFT-based reconstruction, (d) reconstruction based on our non-uniform spline algorithm ($\lambda = 0.000176$).



(a)

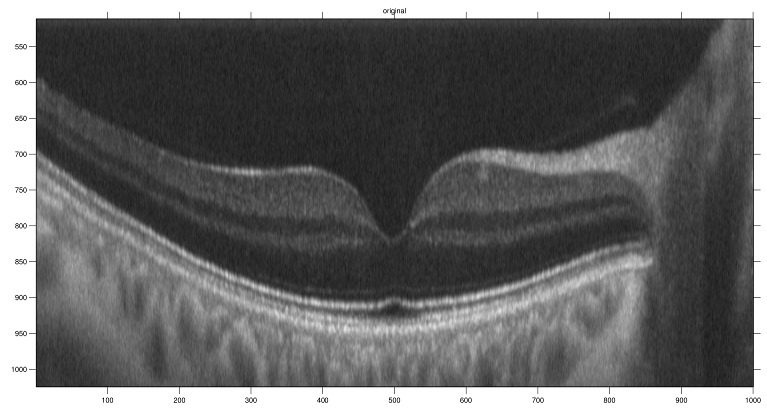


(b)

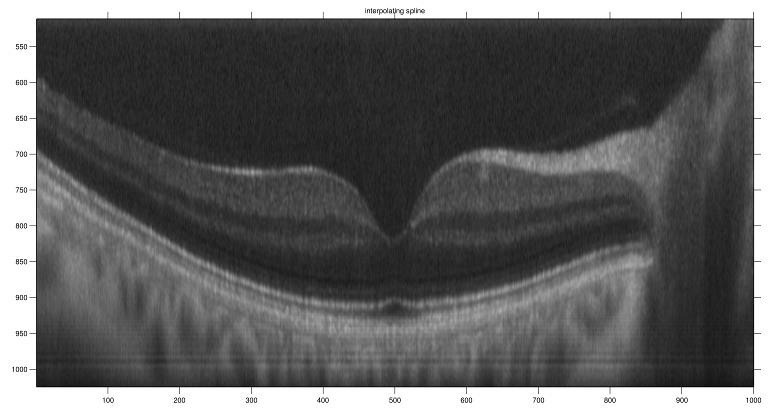


(c)

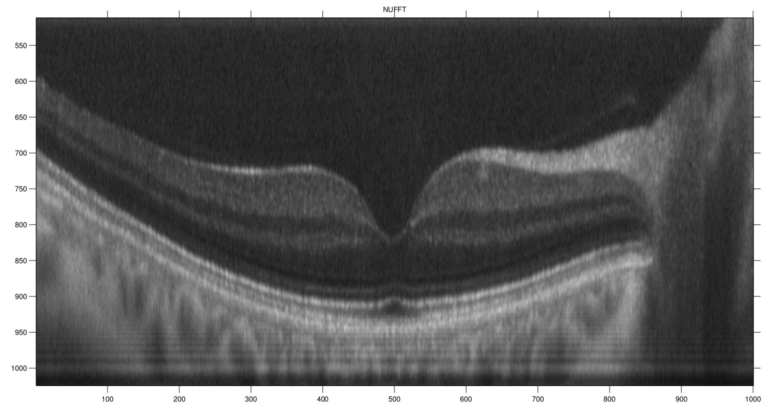
Fig. 4.23: Visualization of the reconstruction error for a synthetic OCT data with a diagonal line pattern. (a) reconstruction error for the interpolating spline-based solution, (b) reconstruction error for the NUFFT-based solution, (c) reconstruction error for the solution based on our non-uniform spline algorithm ($\lambda = 0.000176$). In all cases the error was amplified by 4 relative to the gain used for visualization of the reconstruction results on Fig. 4.22



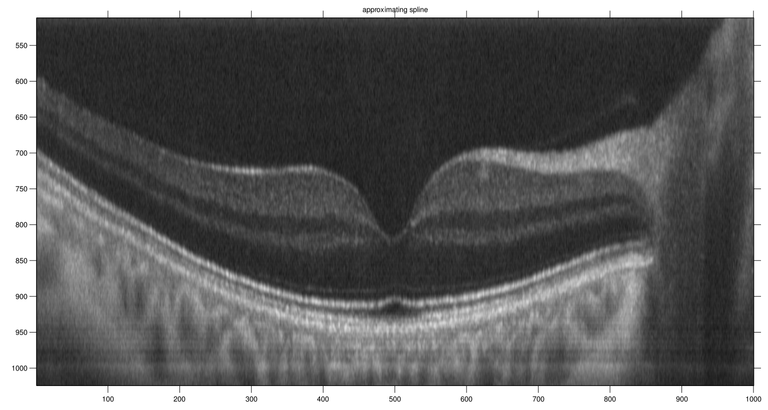
(a)



(b)

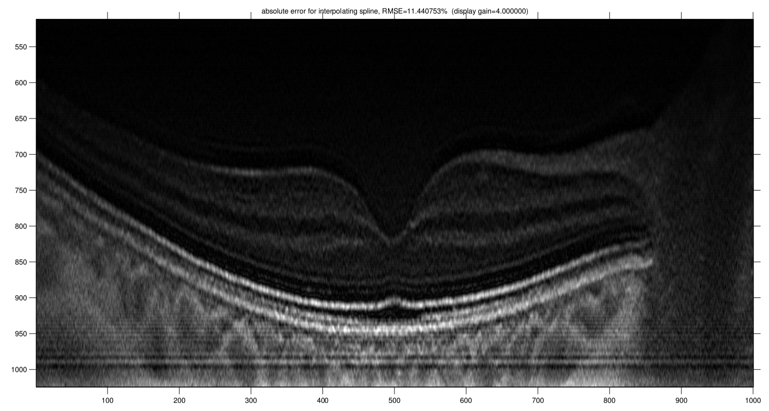


(c)

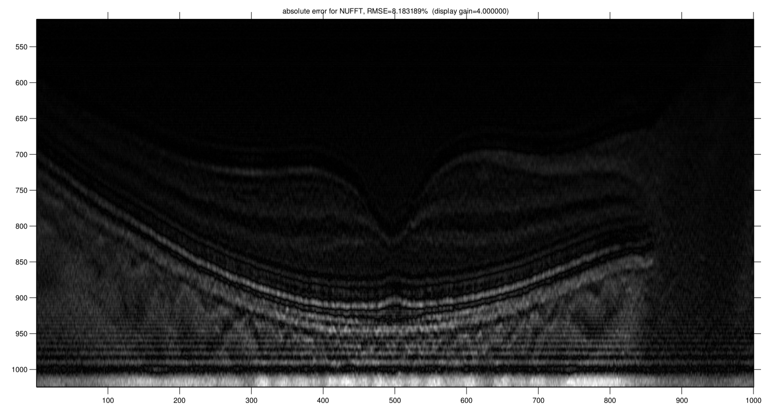


(d)

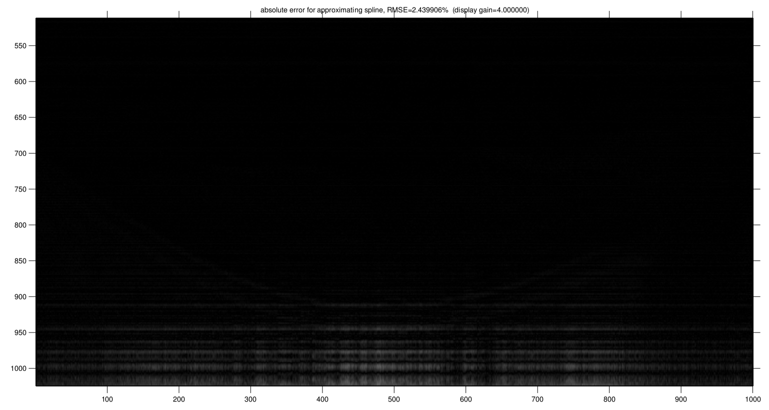
Fig. 4.24: Visualization of the reconstruction of a synthetic OCT data generated based on a high-quality image of retina. (a) the true solution, (b) interpolating spline-based reconstruction, (c) NUFFT-based reconstruction, (d) reconstruction based on our non-uniform spline algorithm ($\lambda = 0.000176$)


absolute error for interpolating spline, RMSE=11.440753% (display gain=4.000000)

(a)


absolute error for NUFFT, RMSE=8.183189% (display gain=4.000000)

(b)


absolute error for approximating spline, RMSE=2.439906% (display gain=4.000000)

(c)

Fig. 4.25: Visualization of the reconstruction error for a synthetic OCT data generated based on a high-quality image of retina. (a) reconstruction error for the interpolating spline-based solution, (b) reconstruction error for the NUFFT-based solution, (c) reconstruction error for the solution based on our non-uniform spline algorithm ($\lambda = 0.000176$). In all cases the error was amplified by 4 relative to the gain used for visualization of the reconstruction results in Fig. 4.24

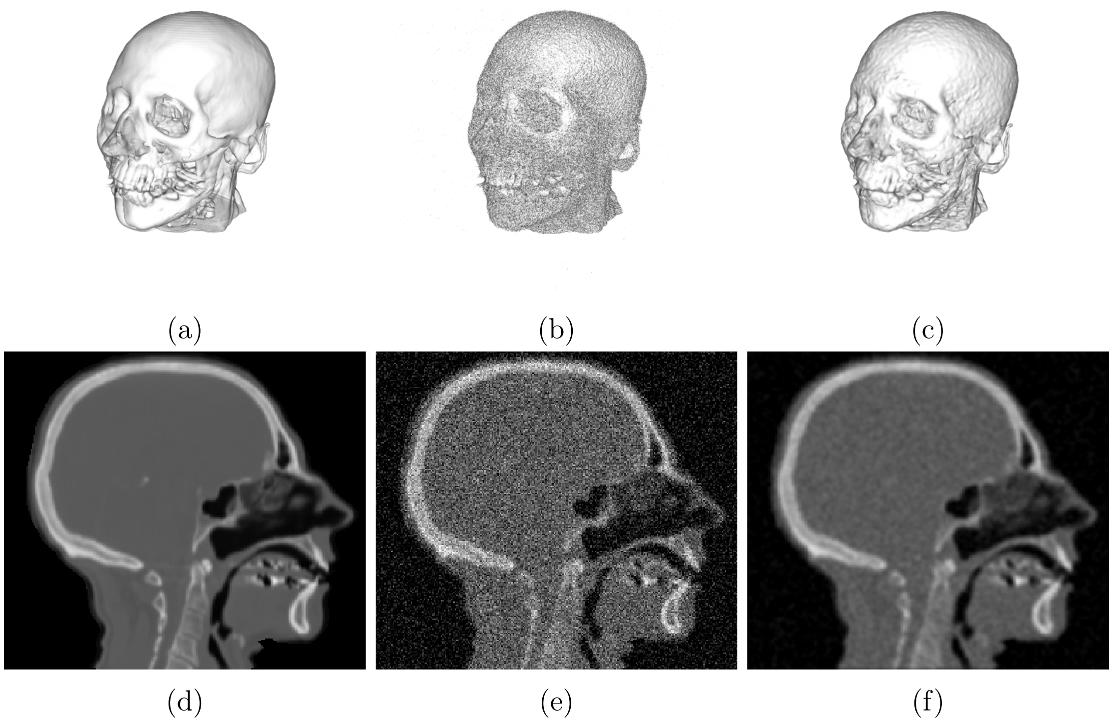
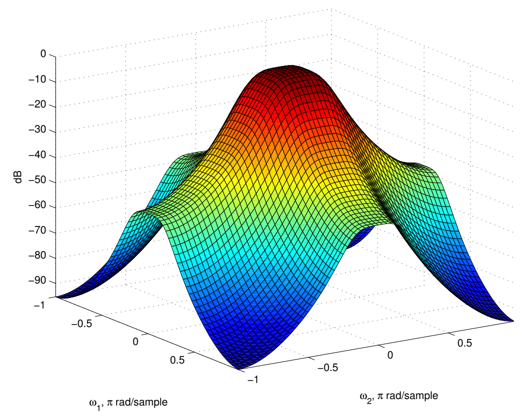
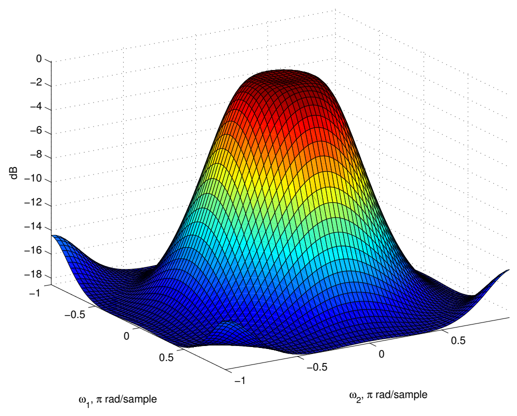


Fig. 4.26: An example of reconstruction of a medical CT dataset contaminated by Gaussian noise with a standard deviation of 30 ($SNR \approx 7.7dB$) using a separable cubic smoothing spline. (a) a 3-D visualization of the original dataset, (b) a 3-D visualization of the dataset contaminated by noise, (c) a 3-D-visualization of the GCV-optimized smoothing spline solution ($SNR \approx 22.46$ dB), (d) a 2-D slice of the original dataset, (e) a 2-D slice of the dataset contaminated by noise (f) a 2-D slice of the smoothing spline solution



(a)



(b)

Fig. 4.27: (a) A comparison of the amplitude frequency responses of separable and non-separable cubic ($n_1 = n_2 = 3$) smoothing spline operators (a) separable smoothing spline with $\lambda = 5$ for both dimensions, (b) non-separable smoothing spline based on Duchon regularizer of order $p = 2$ with $\lambda = 0.5$

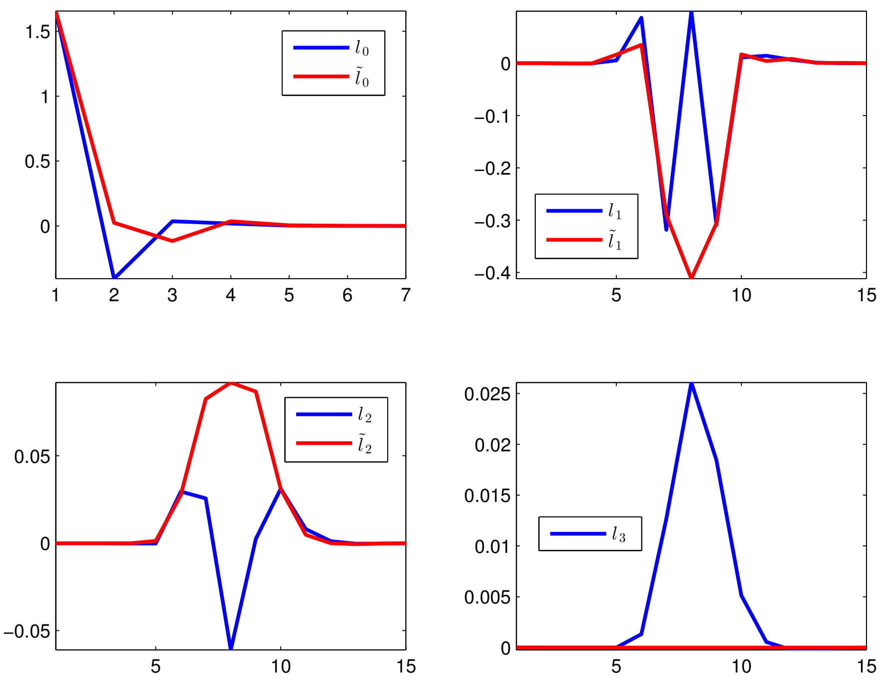


Fig. 4.28: An example of multi-direction decomposition of a non-separable smoothing spline operator constructed based on the tensor product of quadratic and cubic B-splines ($n_1 = 2, n_2 = 3$) and second order Duchon seminorm. One-dimensional filters $\boldsymbol{l}_v, v = 0 \dots n_2$ and $\tilde{\boldsymbol{l}}_v, v = 0 \dots n_1$ were computed by decomposing the tensor filter $A(z_1, z_2)$ along the second and the first dimension respectively.

Chapter 5

High-level FPGA-based system design for high-performance signal reconstruction

In the previous chapters we showed how a detailed analysis of the structure of a spline-based variational formulation of the signal reconstruction problem led to the derivation of highly efficient algorithms. Our practical computational experiments confirmed that the derived algorithms can offer outstanding performance when implemented on the current computing platforms such as multi-core CPU and DSP. The key for achieving that is the highly structured nature of the involved computations, which are due to locality of the used spline bases, their convolution, multiresolution and tensor properties. In this chapter we describe our practical experience with yet another computing technology that allows to create high-performance embedded computing platforms based on the properties of splines and tensors.

5.1 FPGA computing technology

Field Programmable Gate Array (FPGA) are integrated circuits that are based on a large array of programmable logic components connected by configurable interconnects. One of the key features that distinguishes FPGA from other standard computing technologies is that such circuits can be programmed to any desired application or functionality requirements be it a general purpose processor or any custom digital design. In many cases FPGA become a preferable choice for industries due to the fact that they combine the specific advantages of Application Specific Integrated Circuits (ASIC) and processor-based systems. Similarly to ASIC, FPGA offer a hardware-timed speed and reliability but do not require large upfront expenses that is the typical prerequisite for a high-volume production of ASIC devices. FPGA programmable logic has the same flexibility as a software running on a processor-based system but is not limited to the particular architecture and the available number of cores. In contrast to CPU, FPGA are fully parallel devices where the granularity of parallelism can be achieved at an arbitrary level by as-

Table 5.1: Advantages and disadvantages of FPGA in comparison with other computing technologies

Advantages	**Disadvantages**
Performance due to full hardware parallelism FPGA offer more computing power in terms of operations per cycle than CPU and DSP; they are especially powerful for data streaming applications	*Performance* FPGA do not offer the highest possible performance for complex processing algorithms; FPGA provide lower operating frequency than ASIC
Fast time to market FPGA offers flexibility and rapid prototyping capabilities. It is possible to test an idea without manufacturing. FPGA development tools have a great impact on development productivity.	
Cost FPGA do not require large upfront expenses for design and production in contrast to ASIC	*Cost* FPGA are more expensive than ASIC in case of a high volume production
Power consumption FPGA offer higher power efficiency in terms of performance/Watt than CPU and GPU	*Power consumption* currently FPGA consume more power than ASIC
Reliability true parallel "hard" execution of a task without resource sharing and scheduling, absence of OS minimizes reliability concerns in FPGA-based systems	
Maintenance FPGA can be reprogrammed at any time by improving and adding functionalities to a product	

signing a concrete out of many operations to a separate part of FPGA without resource sharing. Of course, FPGA is not an ideal technology for all scenarios and along with its advantages has also disadvantages. Table 5.1 lists the advantages and disadvantages of FPGA technology compared to other existing computing platforms.

5.1.1 Conventional FPGA development approaches

Initially FPGA were used only by hardware engineers who understood in details what is inside of FPGA and how it works. Today, due to a progress in the direction of high-level FPGA development, the situation has improved but not such that the technology becomes easily accessible to software engineers who are not specialized in the field of digital circuit design. A conventional model for FPGA development assumes the use of low-level programming tools that are supported by Hardware Description Languages (HDL) such as VHDL (based on the Ada programming language) and Verilog (a C dialect). These two low level languages, are still the de facto standards used for development of FPGA systems in many industrial fields such as in Communications, System Control, Medical Signal and Image processing systems. Due to the low level nature of HDL languages it can be difficult to use them for implementing complete complex systems. As a side effect this leads to long development cycles and difficulties in maintainability of development projects.

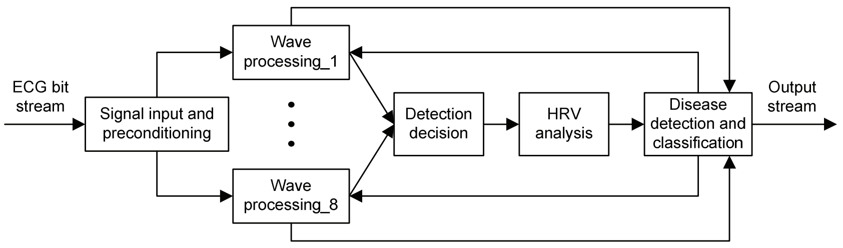
In our joint effort together with Computer System Institute of ETH Zuerich and HighDim GmbH we developed an alternative high-level approach to efficient design of computing systems on FPGA. This approach is mainly targeted towards application specific design of stream-based data processing systems, which is in perfect fit with our needs for signal processing applications. A special feature of the proposed approach is that it allows to design hybrid architectures in a software/hardware co-design manner: a code written by a software engineer is used for generation of both software and hardware parts of the system that are to be deployed and executed on FPGA. This can be achieved with ActiveCells - a high-level programming model and an integrated development environment for hybrid software/hardware co-design of application-specific streaming data processing systems on FPGA, which is described in the next sections.

5.2 ActiveCells: a high-level hardware/software co-design on FPGA

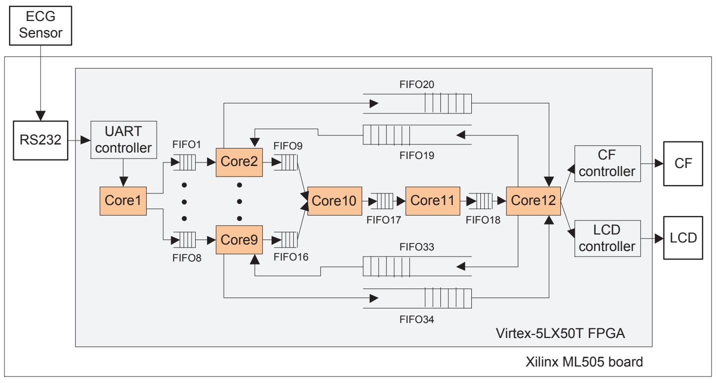
5.2.1 Multi-processor streaming system design

The work on ActiveCells originated from the joint project "Supercomputer in the pocket" of ETH Zuerich and University Hospital of Basel funded by Microsoft Innovation Cluster for Embedded Software (ICES). The project targeted an application from the field of medical signal processing. The achieved results of this development were described in [75, 76]. Basically at that stage the work was based on the concept of software/hardware co-design applied to design of application specific multi-processor streaming systems on FPGA. According to this concept an application is developed as a high-level program written in Oberon programming language. The program describes multiple processes that run a certain activity and communicate with each other using communication channels. In a top-level module the developer can describe the architecture by instantiating the processes, specifying their capabilities such as the size of instruction/data memory, IO peripherals and finally by creating the corresponding interconnects. At the compilation stage the system is translated to FPGA hardware by mapping processes to individual soft-core processors, Tiny Register Machines (TRM) - a minimalistic CPU based on the design of Prof. Niklaus Wirth [77]. The inter-process communication channels is mapped to First Input First Output (FIFO) buffers. An example of an architecture that was built on the base of the described principles is shown in Fig.5.1.

The developed concept showed its usefulness for development of embedded medical



(a)



(b)

Fig. 5.1: An example of multi-processor ECG processing architecture. (a) a block-diagram of the desired application (b) a block-diagram of the resulted computing architecture implemented on FPGA

systems such as the ones that perform ECG or EEG real-time analysis. However, when applied to medical imaging, it faced a natural problem of all processor-based systems. The sequential way of processing within a general purpose processor, the overhead in data communication and the relatively low speed of the processors achievable on FPGA (a TRM can run at a frequency of not more than 125 MHz on a Virtex-5 platform from Xilinx) made the approach practically unusable in terms of achievable computing performance even for simple image processing algorithms. An attempt to solve this problem was made by extending the capabilities of processor cores using the standard concept of Single Instruction Multiple Data (SIMD) parallelism: a vector version of the TRM was implemented [76]. An example of a simple motion detection architecture that implements a motion detection functionality, which was implemented on the base of multiple vector processors is presented in Fig.5.2. While vectorization of data processing helped to increase the speed, it was way far from being high-performance: the maximal frame rate of the motion detection architecture in Fig.5.2 was at about 18 frames per second at the image resolution 576 x 768 pixels. In addition, the increased complexity of the designed vector processor design led to a significant increase of the FPGA resource usage and decrease of the maximal achievable clock frequency. Another problem of the approach, at that initial development stage, was a difficulty in extending the capabilities of the processor components with various types of features e.g. an SPI communication

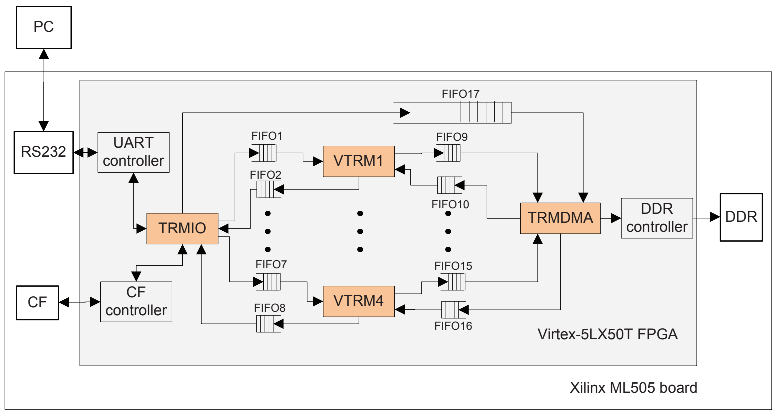


Fig. 5.2: A block diagram for a motion detection architecture built on the base of multiple vector processors

interface or a convolution engine. Such capabilities were treated in a traditional way as peripherals that belong to the processor (similarly to the controllers on the ARM-based platforms). In this case when the developer wanted to integrate some new peripherals he had to recompile the compiler, a requirement that is unacceptable in real industrial development settings.

5.2.2 A hybrid system design

The problem of the multi-processor approach described above was due to the fact that it did not fully exploit the parallelism offered by FPGA. Using a processor with its inherent sequentiality and the associated overhead, it is very difficult to beat the performance of a simple component dedicated to a specific task. A natural solution to overcome the experienced problem is based on a hybrid design approach that combines general purpose processors with their flexibility and reprogrammability and specialized hardware components, which can perform a specific, very structured and parallel task with highest possible performance. The proposed approach was implemented in ActiveCells.

5.2.3 ActiveCells programming model

The ActiveCells programming model was developed by Felix Friedrich at ETH Zuerich [78] as an extension of the Oberon programming language. The main construct that is used for implementing an ActiveCells architecture is a **cell** – a processing element that performs some specific activity. Cells in an architecture can communicate with each other by means of buffered channels and can form a network – a **cellnet**. Listing 5.1 shows a simple example of an architecture that implements filtering of a data stream using a 3-tap symmetric FIR filter. As we see from the example that a cell is defined as a type. The body of a cell defines its activity. In this case a cell works indefinitely in time, which explains the use of an infinite loop.

Cell communication ports

A cell can have multiple input and output ports which are used for communication with other cells in a cellnet. For example an output port `filterCfg` of the cell instance `ctl` is connected to the corresponding input of the cell instance `filter` and is used for runtime configuration of the filter coefficients. The cells communicate by sending and receiving data to/from their ports. Send and receive operations can have a blocking or non-blocking behavior. For example, the code `serInp ? cmd;` corresponds to a blocking receive operation: in this case the cell activity is blocked until a data item becomes available from the port `serInp` and is received to the variable `cmd`. A non-blocking receive is abstracted using operator **??**. If data is available on a specified port, the operator receives the data to a specified memory location and returns a boolean status **true**. In case of data absence the operator returns **false** and does not lead to blocking of the cell's activity. In the presented example the non-blocking receive mechanism is used for runtime reconfiguration of the filter cell. In a similar way the language defines blocking send (operator **!**) and non-blocking send (operator `!!`) operations (Appendix 5.A explains the need for this operation).

```
cellnet SimpleCellnet;
import Engines, Filters;

type

   Controller = cell{Arch="TRM"}(serInp: port in; filterCfg: port out);
   var cmd: integer; h: array [2] of real;
   begin
      loop
         serInp ? cmd; (* blocking receive *)
         h := Filters.GetFilter(cmd);
         filterCfg ! h; (* blocking send *)
      end
   end Controller;

   SymFirFilter3 = cell(cfg, x: port in; y: port out)
   var h: array [2] of real; x0, x1, x2: real; i: integer;
   begin
      h[0] := 4/6; h[1] := 1/6;
      loop
         if (cfg ?? h[i]) then i := (i+1) mod 2; end; (* non-blocking receive *)
         if (x ?? x0) then y ! h[1]*(x0 + x2) + h[0]*x1; x2 := x1; x1 := x0; end;
      end
   end SymFirFilter3;

const
   DefaultBaudrate = 115200;
var
   ctl: Controller;
   uartRx{Baudrate=DefaultBaudrate}: Engines.UartRx;
   filter : SymFirFilter3;
   ioStream{Standard="SingleEnded"}: Engines.SerialIoPort;
begin
   new(ctl); new(uartRx); new(filter); new(ioStream);
   connect(uartRx.output,ctl.serInp,256);
   connect(ctl.filterCfg, filter.cfg);
   connect(ioStream.output,filter.x); connect(filter.y,ioStream.input);
```

```
end SimpleCellnet.
```

Listing 5.1: A simple ActiveCells architecture that implements filtering of a stream of signal samples with a symmetric 3-tap FIR filter

Instantiation of cell architectures

Cells in an ActiveCells architecture are instantiated and connected with each other in the body of a cellnet. This is done by the use of the operators **new** and **connect** as exemplified by the SimpleCellnet architecture. The **connect** statement provides the mean for connecting the corresponding output and input cell ports using buffered channels (FIFO). The developer has the ability to define the depth of a channel (the length of the corresponding FIFO) using the third argument of **connect**. For example, the code **connect**`(uartRx.output,ctl.serInp,256);` will lead to instantiation of a buffered channel with depth of 256 elements that connects the output port `uartRx.output` with the input port `ctl.serInp`.

Cell parametrization

The functionality and capability of cells in ActiveCells can be controlled by means of so called capability parameters. These parameters are static in the sense that their values have an effect only at the time of instantiation of a cell. There are three ways to specify these parameters in an ActiveCells code: 1). in a cell type definition, e.g. as in the case of definition of the cell type `Controller`, where the parameter `Arch` specifies the type of the instruction set architecture that will be used for implementing the cell in FPGA hardware; 2). in variable definition (**var**) section, e.g. `uartRx{Baudrate=115200}: Engines.UartRx;`, where the parameter `Baudrate` determines the baudrate of a UART receiver component; 3). at the time of cell instantiation, e.g. **new**`(uartRx{Baudrate=115200});`.

Cell hierarchies

ActiveCells allows to create hierarchies based on already defined cell types. Listing 5.2 provides an example how a multi-channel FIR filter can be created as a hierarchical structure from a number of instances of cell type `SymFirFilter3` defined in the code from Listing 5.1. The ports of the new cell type `MultiChannelFilter` are delegated to the ports of the internal components using the language construct **delegate**. The cell type `MultiChannelFilter` can now be used for creating another level of hierarchy together with other existing cell and cellnet types. In this way ActiveCells provides the possibility to design architectures of arbitrary complexity.

```
const
   NumChans = 4;
type
   MultiChannelFilter = cellnet(cfg: port in; x: array NumChans of port in; y: array NumChans of port out);
   var
      k: longint;
      filters : array NumChans of SymFirFilter3;
   begin
```

```
    for k := 0 to len( filters )−1 do
        new(filters [k]);
        delegate(cfg, filters [k]. cfg );
        delegate(x[k], filters [k]. x);
        delegate(y[k], filters [k]. y);
    end;
end MultiChannelFilter;
```

Listing 5.2: An example of creating cell hierarchies in ActiveCells

5.2.4 Communication protocol

As in any computing network the communication between cells in an ActiveCells architecture requires a protocol. It is desirable that such a protocol is generic enough to allow a unified plugability of the components and at the same time provides high data throughput, which is especially critical for signal and image processing systems. In consideration of these two important aspects we decided to choose the AXI4-Stream industrial standard [79] as a protocol for inter-cell communication. AXI4-Stream was designed by ARM for high-speed streaming-based systems. It is a subset of a more general AMBA AXI4 memory-mapped protocol. To guarantee a consistent data transfer from a master (a component that initiates the transfer) to a slave (a component that accepts the data) AXI4-Stream defines a minimal set of three signals: **TVALID** signal that is asserted by the master to indicate data availability, **TREADY** signal that is controlled by the slave to inform the master about readiness to accept data and **TDATA** – the actual data signal driven by the master. At the hardware level an AXI4-Stream data transaction is assumed to be accomplished when at the current clock cycle both **TVALID** and **TREADY** are asserted by the master and the slave correspondingly. The timing diagram in Fig. 5.3 exemplifies the way how the protocol works by four possible situations:

- time points A: no transfer takes place because there is no data available from the master and the slave is not ready
- time points B: the slave is ready to accept new data but there is no data available from the master (Master Waitstate)
- time points C: there is data available from the master but the slave is not ready; the master has to hold the current data until a transaction takes place
- other time points: transactions take place and the data is changed by the master at every clock cycle

AXI4-Stream also provides a mechanism for routing (multiplexing) within a computing network. A minimal routing functionality can be implemented using the signal **TDEST**. On both the master and the slave sides this signal has output direction. The master uses this signal to select the slave to whom the the data and its availability are directed. Similarly, for the slave this signal provides the possibility to select a master from which the data is to be read.

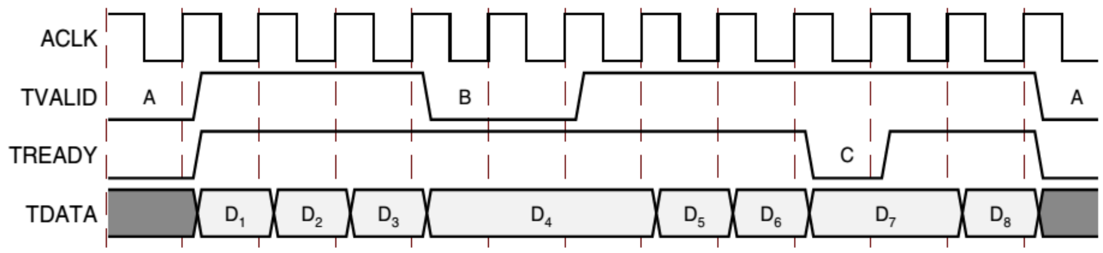


Fig. 5.3: A timing diagram explaining the data transactions in AXI-4 stream communication protocol

The protocol defines other optional signals that can be used for implementing various functionalities e.g. merging of data streams, their up-sizing, down-sizing etc. In general, the protocol provides a high level of flexibility and can satisfy the needs of many applications. In addition to that, a fundamental advantage of AXI4-Stream protocol is that the data transfers can take place at every cycle with minimal overhead in terms of the resources used for implementing the protocol in FPGA hardware. All other optional features of the protocol come at the resource usage cost rather than at the cost of performance loss.

We used the AXI4-Stream standard for implementing inter-cell communication at the hardware library level of ActiveCells. Cell output and input ports were implemented in FPGA as master and slave AXI4-Stream ports correspondingly. As a basic set of signals we used **TDATA**, **TVALID** and **TREADY** described above. The routing functionality provided by **TDEST** signal is mainly supported by processor components. Other signals defined by the protocol are implemented with optional support.

Adoption of the AXI4-Stream protocol into the ActiveCells framework was a big asset. The protocol greatly facilitated the automatic FPGA hardware generation based on high-level source code and it allowed us to unify the way how hardware components are implemented and integrated into ActiveCells hardware libraries (as will be explained below). At the same time its efficiency and minimal redundancy permitted the development of high-performance streaming data processing systems.

5.2.5 ActiveCells hardware components: Engines

If we compile the example architecture from Listing 5.1 using the ActiveCells compiler, the instance `filter` of cell type `SymFirFilter3` will be implemented in FPGA hardware as a separate processor. This processor will execute the binary machine code corresponding to the high-level code within the body of the cell type. In case if the data stream at the input port `filter.x` comes at a high speed (e.g. at the clock frequency of the processor) the component will not be able to provide the required performance due to sequentiality of code execution and the memory access overhead. Thus, we meet exactly the same problem of the multiprocessor approach described above. Now, assume that in a hardware library available to the ActiveCells compiler there is a dedicated hardware component (Engine) with name `SymFirFilter3`. In this case the developer can request from the compiler to replace the processor by that specialized hardware component. This can be done using a built-in parameter `Engine`. For example, in the particular case of `SymFirFilter3` cell type it is sufficient to specify this parameter at instantiation time: **`new`**`(filter{Engine});`. The just described mechanism allows to create hybrid software/hardware architectures by combining the flexibility of processor-based systems and high performance of dedicated hardware components.

Engines in ActiveCells hardware libraries are implemented based on HDL languages. This can be done either manually by an experienced hardware engineer or using automated tools (e.g. CORE Generator from Xilinx). The interface of the corresponding HDL code has to obey the communication protocol described above. For example, a hardware component corresponding to the cell type `SymFirFilter3` could have an interface in Verilog HDL as shown in Listing 5.3.

```
module SymFirFilter3
(
    input aclk,
    input aresetn,

    input [31:0] cfg_tdata,
    input cfg_tvalid,
    output cfg_tready,

    input [31:0] x_tdata,
    input x_tvalid,
    output x_tready,

    output [31:0] y_tdata,
    output y_tvalid,
    input y_tready
);
```

Listing 5.3: Interface example of an ActiveCells hardware component implemented in Verilog HDL

Provided that HDL source code of a hardware component is already available, the component can be conveniently integrated into ActiveCells hardware library. The only requirement is a description of this component in a generic specification file. An example of such a representation is presented in Appendix 5.B. A specification provides the compiler with all necessary information for automatic hardware generation. Namely:

- *software/hardware port mapping*: information about the correspondence of ports in the hardware implementation to the logical ports of a respective cell. For example, in the `SymFirFilter3` component the logical port `x` has in correspondence an AXI4Stream port with same name and respective attributes. This information is used for automatic connectivity of the hardware components during the hardware generation stage.

- *supported FPGA platforms*: a sufficiently general naming scheme allows to inform the compiler about the supported target devices for a component. This parameter allows the compiler to choose a suitable implementation among possibly multiple implementations of the same component.

- *source code dependencies*: a list of all files required for the process of hardware implementation.

- *component parameters*: provision of the correspondence of cell capability parameters with their default values to the respective HDL parameters of the hardware component. In order to provide the necessary flexibility, this feature in the compiler is supported by code interpretation.

Even processor cores are described in the same manner and, therefore, also treated as hardware components. With this unified component-based approach ActiveCells hardware libraries can be conveniently and flexibly enriched by new types of processing units.

Once the steps above have been carried out the component becomes automatically available to the ActiveCells toolchain. At this point the work of a hardware engineer who implements the hardware component is finished and a software engineer can take over the work on implementing the required architecture using high level tools without further involvement of the hardware engineer.

However, in order to achieve high performance it is not necessary to implement hardware HDL components for every performance critical task. Sometime it suffices to rely on a limited number of generic building blocks – high-performance hardware components that are used for generation of components with higher complexity. This is made possible by the cell hierarchies as described on page 89.

Listing 5.4 presents an example how the `SymFirFilter3` component can be implemented at a high level using generic primitives available in the ActiveCells hardware library. A conventional block-diagram representation of the FIR filter implemented by the high-level component `SymFirFilter3` and its corresponding block-diagram generated directly from the code using ActiveCells toolchain are shown in Fig.5.4. Note, that high-level components implemented in this way by a software engineer can provide the performance comparable to that of hardware components implemented by an experienced hardware engineer directly in HDL code.

```
type

SymFirFilter3 = cellnet(cfg: port in; x: port in; y: port out);
var
    cfgInterleave : Engines.StreamReshaper1x2;
   mul: array 2 of Engines.MulFlt32;
   add: array 2 of Engines.AddFlt32;
   delay, h: array 2 of Engines.StreamRegister;
   k: integer;
begin
   for k := 0 to 1 do
      new(delay[k]{Preloaded=true,PreloadValue=0});
      new(mul[k]);
      new(add[k]);
      new(h[k]{Preloaded=true,PersistentOutput=1});
   end;

   (* interleave  coefficients  configuration stream *)
   new(cfgInterleave{InitN=2});
   delegate(cfg,cfgInterleave.input);
   connect(cfgInterleave.output[0],h[0].input,0);
   connect(cfgInterleave.output[1],h[1].input,0);
```

```
    (* delay chain: x2 := x1; x1 := x0; *)
    delegate(x,delay[0].input);
    connect(delay[0].output,delay[1].input,0);

    (* u := h[1]*(x0 + x2) *)
    connect(h[1].output,mul[0].input[0],0);
    delegate(x,add[0].input[0]);
    connect(delay[1].output,add[0].input[1],0);
    connect(add[0].output,mul[0].input[1],0);

    (* v := h[0]*x1 *)
    connect(h[0].output,mul[1].input[0],0);
    connect(delay[0].output,mul[1].input[1],0);

    (* y := u + v *)
    connect(mul[0].output,add[1].input[0],0);
    connect(mul[1].output,add[1].input[1],0);

    delegate(y,add[1].output);
end SymFirFilter3;
```

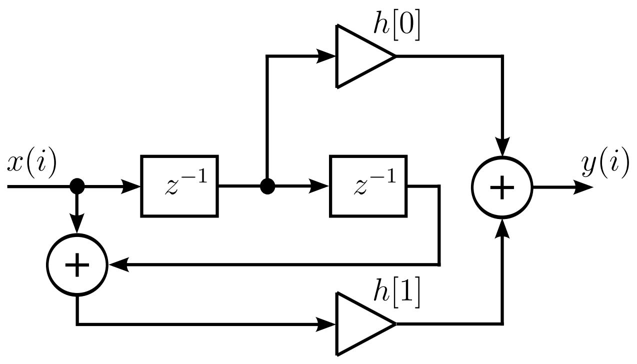
Listing 5.4: SymFirFilter3 component from Listing 5.1 hierarchically constructed from generic building blocks

A multi-channel version of the filter component from Listing 5.4 demonstrates the high performance capability of FPGA in an impressive way: we deployed a total amount of 26 channels of the symmetric 3-tap FIR filter on a low-end Xilinx Zynq XC7Z020-1 FPGA device. The maximal achieved clock rate was 143 MHz. Therefore, the maximal performance provided by this architecture in terms of floating point operations per second (FLOPS) is 26 channel * 4 operations * $143 \times 10^6 \approx 14.87$ GFLOPS. This approximately corresponds to performance of a high-end Intel i7 Quad-core processor with clock frequency of $\approx$ 3 GHz. Taking into account that for the same workload the Intel processor requires a power budget of at minimum an order of magnitude higher than the used FPGA device, we clearly see the advantage of using FPGA. This example also shows the benefits of high-level component-based ActiveCells design.

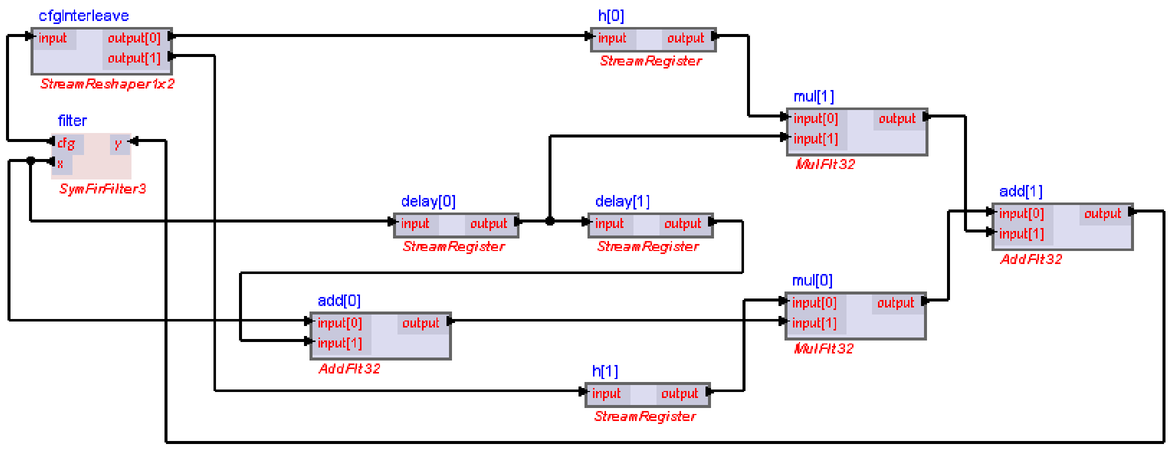
5.2.6 ActiveCells target device specification

At the time of compilation of an ActiveCells architecture it is required to provide a specification of a concrete target device where the architecture will be deployed. This assumes the provision of the following information to the compiler:

- *FPGA part specification*: all part-related information including part name, package type, speed grade etc.
- *system signals specification*: system clock and system reset signals together with their respective parameters and pins specification; possibly other clock signals provided by the target device.
- *terminal port specification*: an ActiveCells hardware architecture normally exposes some of its signals to the outside world via FPGA device pins. This exposition is



(a)



(b)

Fig. 5.4: (a) a conventional FIR-filter block-diagram representing functionality implemented by the `SymFirFilter3` component from Listing 5.4. (b) a block-diagram automatically generated using the ActiveCells toolchain from the high-level code of this component.

done via so called terminal ports that are defined in hardware component specifications. In this sense the task of the target specification is to provide the correspondence between the terminal port signals and the respective FPGA pins, including the direction of the signals, IO standards etc.

Similarly to hardware component specifications target devices are also specified using a generic specification and therefore can be flexibly integrated into the ActiveCells hardware libraries. An example of a target device based on Xilinx Spartan-6 FPGA is presented in Appendix 5.C.

5.2.7 Model-based system design

In this paragraph we shortly comment on ActiveCells as a tool for model-based system design.

The conventional approach to the design of digital systems assumes a separation in the development process that is usually manifested by division of the development team into a software and a hardware group, which work on respective software and hardware parts of the system. While being somehow related to the strategy "divide and conquer" this approach can introduce serious difficulties especially in the case of designing complex systems. The mentioned separation can make the understanding of the whole system more difficult, it can increase the complexity of verification, testing, documentation and maintaining of the system design. This in turn introduces a risk to make the development longer and to adversely impact the time-to-market for the corresponding product.

In contrast to the traditional approach model-based design [80] is built on the principle of unification of the development processes. In this approach a model of the whole system is the key. It plays a role of an executable specification that is continuously refined throughout the development process. Significant advantages of model-based design include the fact that it facilitates rapid design and it moves the verification process all the way to the beginning of the design cycle. This helps to detect system specification related errors, design errors, and implementation errors early.

In the context of embedded system design the model-based approach is supported by the following means:

- common design environment used by all members of the development team
- ability to test and verify the whole system and its functionalities via system model simulation
- automatic generation of software and hardware releases of the system
- unified way of documenting the system

The proposed concept of ActiveCells system development is actually in line with the model-based approach. It does offer a unified design environment for hardware/software co-design with consideration of a system as a whole entity. It does allow automatic generation of the software and hardware parts of the system for deployment on a target. Just recently we introduced (thanks to Dmytro Shulga at University Hospital of Basel) a possibility to test and verify the correctness of a designed ActiveCells architecture by software-supported simulation of the corresponding system model. The ActiveCells systems documentation process is facilitated by the tools developed by Felix Friedrich at ETH Zuerich that allow to document the projects directly in the code.

5.2.8 ActiveCells design flow

After description of the concepts and generic mechanisms used by ActiveCells we would like to conclude with a general description of the design flow in ActiveCells. At the origin of a design is a source code developed by a software engineer. This source code is first processed by the compiler frontend that basically generates a high-level syntax tree, which describes the architecture. After this the design flow can have three possible directions (see the diagram presented on Fig. 5.5). The developer can first test the functionality of the developed architecture using modeling facilities implemented in ActiveCells (green boxes of the diagram). When, via an iterative process of trial-error-modification steps the design reaches the state of being correct and fully verified at the model level, the developer can initiate the process of hardware generation (the "red" path of the design flow diagram). In this case the respective part of the ActiveCells toolchain processes the previously generated syntax tree of the architecture and based on the available hardware libraries generates an HDL description of the hardware. After that the produced hardware description is used for implementation of real hardware. The duration of the implementation process mainly depends on the complexity of the developed architecture and on the performance of the PC and can span from minutes to hours. A successful completion of the process results in a binary code that describes the hardware configuration of the architecture on FPGA. In case if the architecture contains processing cores the "blue" path of the design flow diagram is activated. An intermediate code corresponding to the software part of the ActiveCells code is generated. The intermediate code is then processed by the respective backend (depending on ISA, e.g. TRM, VTRM, ARM etc.) that generates processor-specific binary instruction and data code. This code is then used for patching the corresponding parts of the FPGA binary code. At this point the architecture can be deployed to the target. This is a relatively quick process that usually takes from few seconds to a dozen of seconds depending on the size of FPGA binary code and on the speed of FPGA configuration interface.

When the developer needs to modify the software-related part of the architecture, the deployment of the modified design does not require the "red" path of the design flow diagram. In this case the already generated FPGA binary code can be reused. This dramatically shortens the development time in ActiveCells and therefore implies a substantially decreased time-to-market for a product under development.

The developer has the ability to document the ActiveCells design directly in the code comments using a special markup language. By request of the developer the compiler frontend will automatically generate an HTML (or possibly another format depending on availability of documentation plugins) document based on the documentation code extracted from the sources ("yellow" path of the design flow diagram).

5.2.9 FPGA Systems on Chip (SoC)

FPGA vendors now offer a new type of devices that integrate an ARM multi-core processing system, with its respective peripherals and memory interfaces, together with conventional FPGA logic tightly connected with the ARM processor via high-bandwidth interfaces. This approach provides multiple benefits in terms of increased overall system performance, increased level of integration, reduced cost and significantly reduced power consumption. To take the advantage of this new technology for implementing high performance signal and image processing systems we integrated into ActiveCells a family of SoCs Zynq-7000 from Xilinx. This integration was facilitated by our flexible component-based representation of processing systems. The Zynq ARM processor is sup-

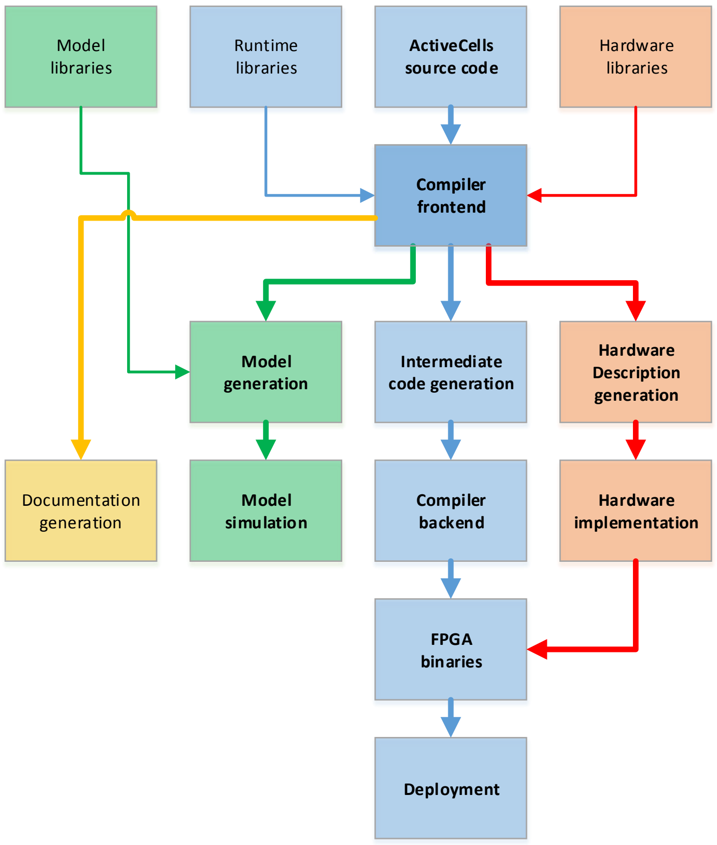


Fig. 5.5: A diagram describing the design flow in ActiveCells

ported in ActiveCells hardware library in the same way as our custom TRM processor. The communication of the processor with other components implemented directly in the FPGA logic is performed using the same AXI4-Stream protocol. For that we had to implement a lightweight converter of the more generic and more complex AXI4 memory mapped interface [81] to AXI4-Stream, which is in fact a subset of the AXI4 standard. The software-related part of ActiveCells development on Zynq devices is supported by the ARM backend of the compiler, Minos operating system [82] and an ARM port of AOS operating system [83].

5.2.10 Multi-FPGA system design

The problems of multidimensional medical signal processing with their large computational requirements might profit from a coarser grained parallelism offered by multi-FPGA architectures. Due to genericity of the used communication interface ActiveCells in its current state can be already used for designing multi-FPGA systems. An example for that is in Listing 5.1, which was presented during the introduction of the ActiveCells programming model. There the cell instance `filter` receives signal data from the output of an instance `ioStream` of type `SerialIoPort` and sends filtered data to the corresponding input of the same cell. `SerialIoPort` is implemented as a hardware component that provides serial connectivity between the FPGA where the `SimpleCellnet` architecture is deployed and any external device, e.g. another FPGA. Currently such development assumes a manual distribution of the task performed by an architecture among the available FPGA devices. Within our ongoing research projects we are working on an extension of the ActiveCells programming model that would allow more flexible development of multi-FPGA architectures with automated resource and task distribution.

5.3 High-performance computational kernel for multidimensional reconstruction on FPGA

In Chapter 3 we proposed a highly efficient iterative tensor decomposition-based algorithm for solving the problem of large multidimensional image reconstruction. At the core of this algorithm is fast computation of the gradient of the underlying cost function. The most computationally challenging part of this computation corresponds to the least-squares term of the cost function. Previously we showed that in 3-D it can be computed based on the following tensor decomposition:

$$\mathcal{V}^{x_1y_1z_1} = \mathcal{U}^{x_1y_1z_1} + (\mathcal{E}^{x_1}\cdot\mathcal{E}_x)\cdot(\mathcal{G}^{y_1}\cdot\mathcal{G}_y)\cdot(\mathcal{H}^{z_1}\cdot\mathcal{H}_z)\cdot\mathcal{C}^{xyz} \tag{5.1}$$

where $\mathcal{E},\mathcal{G},\mathcal{H} \in R^{n+1}$,$\mathcal{C},\mathcal{U},\mathcal{V} \in R^{n+1}\times R^{n+1}\times R^{n+1}$, n is degree of B-spline used for representation of the signal to reconstruct. Let us consider how we could implement this computational kernel of FPGA using ActiveCells.

First we rearrange the terms in equation (5.1) using the properties of commutativity and associativity of the tensor product:

$$\mathcal{V}^{x_1y_1z_1} = \mathcal{U}^{x_1y_1z_1} + \Big(\mathcal{E}_x\cdot\big(\mathcal{G}_y\cdot(\mathcal{H}_z\cdot\mathcal{C}^{xyz})\big)\Big)\cdot\big((\mathcal{E}^{x_1}\cdot\mathcal{G}^{y_1})\cdot\mathcal{H}^{z_1}\big) \tag{5.2}$$

where the term $\mathcal{E}_x\cdot\big(\mathcal{G}_y\cdot(\mathcal{H}_z\cdot\mathcal{C}^{xyz})\big)$ assumes a successive contraction along the dimensions of tensor $\mathcal{C}$ and results in a scalar, while the term $(\mathcal{E}^{x_1}\cdot\mathcal{G}^{y_1})\cdot\mathcal{H}^{z_1}$ is a pure tensor product. In MathOberon language [63] the tensor expression (5.2) can be implemented as shown in Listing 5.5. The proposed computation algorithm requires in total $2n^3+8n^2+12n+6$ multiplications and $2n^3+6n^2+6n+1$ additions. It can be verified that this algorithm offers the minimal possible amount of operations for computing the expression (5.2).

```
procedure TensorKernel(
                        const c, u: array [n+1,n+1,n+1] of real;
                        e, g, h: array [n+1] of real;
                        var v: array [n+1,n+1,n+1] of real
                        );
var
   cxy: array [n+1,n+1] of real;
   w: real;
begin
```

```
    (* perform successive contraction *)
    for x := 0 to n do
        cxy[x ,..]  := c[x  ,..,..]   * h;
    end;
    w := (cxy * g) +* e;

    (* tensor product *)
    v := u + ((w*e) ** g) ** h;
end TensorKernel;
```

Listing 5.5: An example of a MathOberon-based implementation of the tensor computation described by equation (5.2)

The proposed algorithm was implemented in ActiveCells on Xilinx Kintex-7 chip XC7K325T-1FFG676. The developed code together with some block-diagrams automatically generated from it are presented in Appendix 5.D. The implemented design requires about 60% of logic resources available on the particular FPGA device. The maximal achievable speed of the designed architecture for cubic B-spline is 121 MHz. This corresponds to the performance of about 36 GFLOPs.

5.4 B-spline-based non-uniform signal reconstruction on FPGA

In Chapter 4 we presented an algorithm for the reconstruction of one-dimensional signals from arbitrarily sampled measurements. We showed that the algorithm offers an excellent computational performance when implemented on a multi-core CPU-based platform. We implemented the proposed algorithm using ActiveCells and successfully used it for a real life industrial application – medical OCT imaging.

On a low-end Xilinx Zynq XC7Z020-1 FPGA device we could deploy a design with 4 channels of the implemented component running at the clock frequency of 77 MHz. The block size of processed data was 2048 samples. The achieved clock rate corresponds to about 308 MSamples/s – approximately the maximal speed we achieved with our multi-core CPU-based implementation presented in Chapter 4. Taking into account a significantly lower power consumption of the particular configurable target device, we see the advantage of using the FPGA technology. The fact that such a spline-based component can be developed and tested using ActiveCells within a few days – significantly amplifies this advantage.

5.A Non-blocking send in ActiveCells

The need for blocking send operation can be explained using an example code from Listing 5.6. In this example

```
type
Cell = cell(dataInp: port in; dataOut: port out)
var
   rawData: char; (* raw data received from dataInp *)
   buf: array [256] of char; (* buffer with processed data sent to dataOut *)
   inpPos, outPos: integer; (* buffer in/out pointers *)
loop
   (* receive raw data, process it, and fill in processed data buffer *)
   if (dataInp ?? rawData) then
      buf[inpPos] := ProcessData(rawData);
      inpPos := (inpPos+1) mod len(buf);
   end;
   (* send data to a slow communication interface (e.g. UART) *)
   if (dataOut !! buf[outPos]) then outPos := (outPos+1) mod len(buf); end;
   DoOtherTimeCriticalTasks;
end
```

Listing 5.6: An example code explaining the need for non-blocking send functionality in ActiveCells

5.B ActiveCells hardware component specification

```
<component type="HdlComponent" name="SymFirFilter3" hdlModuleName="SymFirFilter3" isa="none">

   <supported>
      <element type="StringValue" value="XC7*"/>
   </supported>

   <ports>
      <element type="ClockHdlPort" name="aclk" mapped="systemClock" direction="in" width="1">
      </element>

      <element type="HdlPort" name="aresetn" mapped="systemReset" direction="in" width="1"
         signalPolarity="false"/>

      <element type="AXI4StreamPort" name="cfg" direction="in" width="32">
         <tdata name="cfg_tdata" mapped="tdata" direction="in" width="32"/>
         <tvalid name="cfg_tvalid" mapped ="tvalid" direction="in" width="1"/>
         <tready name="cfg_tready" mapped ="tready" direction="out" width="1" />
      </element>

      <element type="AXI4StreamPort" name="x" direction="in" width="32">
         <tdata name="x_tdata" mapped="tdata" direction="in" width="32"/>
         <tvalid name="x_tvalid" mapped ="tvalid" direction="in" width="1"/>
         <tready name="x_tready" mapped ="tready" direction="out" width="1" />
      </element>

      <element type="AXI4StreamPort" name="y" direction="out" width="32">
         <tdata name="y_tdata" mapped="tdata" direction="out" width="32"/>
         <tvalid name="y_tvalid" mapped ="tvalid" direction="out" width="1"/>
         <tready name="y_tready" mapped ="tready" direction="in" width="1" />
      </element>
   </ports>

   <parameters>
```

```
        <element type="HdlParameter" name="h0">
            <value type="RealValue" value="?{instance.capabilityParameters['h0'].real:4/6}?"/>
        </element>
        <element type="HdlParameter" name="h1">
            <value type="RealValue" value="?{instance.capabilityParameters['h1'].real:1/6}?"/>
        </element>
    </parameters>

    <dependencies>
        <element type="HdlDependency" fileName="SymFirFilter3.v"/>
    </dependencies>

</component>
```

Listing 5.7: An example of ActiveCells hardware component specification

5.C ActiveCells target device specification

```
<device type="TargetDevice" name="AVSP6LX75T">
    <pldPart type="PldPart" vendor="Xilinx" family="Spartan6" device="XC6SLX75T"
        package="FGG676" speedGrade="−3"/>
    <systemClock type="DerivedClock" name="systemClock" mulRatio="6" divRatio="12">
        <inputClock type="ExternalClock" name="systemClockSrc" frequency="100000000" dutyCycle="50">
            <inputPort type="TerminalHdlPort" name="systemClockSrc" direction="in" width="1">
                <pins>
                    <element name="systemClockSrc" loc="T3"/>
                </pins>
            </inputPort>
        </inputClock>
    </systemClock>
    <systemReset type="TerminalHdlPort" name="systemResetSrc" signalPolarity="TRUE"
        direction="in" width="1">
        <pins>
            <element name="systemResetSrc" loc="AA23" ioStandard="LVCMOS25" pullUp="FALSE"
                pullDown="FALSE"/>
        </pins>
    </systemReset>
    <terminalPorts>
        <element type="TerminalHdlPort" name="UartTxd0" direction="out" width="1">
            <pins>
                <element name="UartTxd0" loc="N20" ioStandard="LVCMOS25" pullUp="FALSE"
                    pullDown="FALSE"/>
            </pins>
        </element>
        <element type="TerminalHdlPort" name="UartRxd0" direction="in" width="1">
            <pins>
                <element name="UartRxd0" loc="N19" ioStandard="LVCMOS25" pullUp="FALSE"
                    pullDown="FALSE"/>
            </pins>
        </element>
    </terminalPorts>
</device>
```

Listing 5.8: An example of ActiveCells target device specification

5.D ActiveCells implementation of a computational kernel used in B-spline based multidimensional signal reconstruction

```
cellnet NusiFpga;

import
    Engines;

const
    n = 1; (* B−spline degree *)
    Dim1d = n + 1;

type
    MulType = Engines.MulFlt32;
    AddType = Engines.AddFlt32;

    (* Tensor product of two 1−D tensors *)
    TensorProduct2d = cellnet(
                                    x: array Dim1d of port in;
                                    y: array Dim1d of port in;
                                    xy: array Dim1d of array Dim1d of port out
                                    );
    var
        mul: array Dim1d of array Dim1d of MulType;
        i, j: longint;
    begin
        for i := 0 to len(mul,0)−1 do
            for j := 0 to len(mul,1)−1 do
                new(mul[i,j]);
                delegate(x[i],mul[i,j].input[0]);
                delegate(y[j],mul[i,j].input[1]);
                delegate(xy[i,j], mul[i,j].output);
            end;
        end;
    end TensorProduct2d;

    (* Tensor product of three 1−D tensors *)
    TensorProduct3d = cellnet(
                                    x: array Dim1d of port in;
                                    y: array Dim1d of port in;
                                    z: array Dim1d of port in;
                                    xyz: array Dim1d of array Dim1d of array Dim1d of port out
                                    );
    var
        tprod2d: TensorProduct2d;
        mul: array Dim1d of array Dim1d of array Dim1d of MulType;
        i, j, k: longint;
    begin
        new(tprod2d);
        for i := 0 to len(x,0)−1 do delegate(x[i],tprod2d.x[i]); end;
        for i := 0 to len(y,0)−1 do delegate(y[i],tprod2d.y[i]); end;

        for i := 0 to len(mul,0)−1 do
            for j := 0 to len(mul,0)−1 do
                for k := 0 to len(mul,0)−1 do
                    new(mul[i,j,k]);
                    connect(tprod2d.xy[i,j],mul[i,j,k].input[0],0);
                    delegate(z[k],mul[i,j,k].input[1]);
                    delegate(xyz[i,j,k], mul[i,j,k].output);
                end;
            end;
        end;
    end TensorProduct3d;

    (* Summation of elements of a 1−D tensor *)
    Sum1d = cellnet(x: array Dim1d of port in; y: port out);
    var
        add: array Dim1d−1 of AddType;
        i: longint;
    begin
        for i := 0 to len(add,0)−1 do
            new(add[i]);
            delegate(x[i+1],add[i].input [1]);
        end;
        delegate(x[0],add[0].input [0]);
        for i := 0 to len(add,0)−2 do
            connect(add[i].output,add[i+1].input [0],0);
        end;
        delegate(y,add[len(add,0)−1].output);
    end Sum1d;

    Contract3d1d = cellnet(
                                    x: array Dim1d of array Dim1d of array Dim1d of port in;
                                    y: array Dim1d of port in;
                                    z: array Dim1d of array Dim1d of port out
                                    );
    var
        mul: array Dim1d of array Dim1d of array Dim1d of MulType;
        i, j, k: longint;
        sum1d: array Dim1d of array Dim1d of Sum1d;
    begin

        for i := 0 to len(mul,0)−1 do
            for j := 0 to len(mul,1)−1 do
                for k := 0 to len(mul,2)−1 do
                    new(mul[i,j,k]);
                    delegate(x[i,j,k], mul[i,j,k].input [0]);
                    delegate(y[k],mul[i,j,k].input [1]);
                end;
            end;
        end;

        for i := 0 to len(sum1d,0)−1 do
            for j := 0 to len(sum1d,1)−1 do
                new(sum1d[i,j]);
                for k := 0 to len(mul,2)−1 do
                    connect(mul[i,j,k].output,sum1d[i,j].x[k],0);
                end;
                delegate(z[i,j], sum1d[i,j].y);
            end;
        end;
    end Contract3d1d;

    Contract2d1d = cellnet(
                                    x: array Dim1d of array Dim1d of port in;
                                    y: array Dim1d of port in;
                                    z: array Dim1d of port out
                                    );
    var
        mul: array Dim1d of array Dim1d of MulType;
        i, j, k: longint;
        yy: array Dim1d of Engines.Fifo;
        sum1d: array Dim1d of Sum1d;
    begin

        for i := 0 to len(yy,0)−1 do
            new(yy[i]{Length=0}); delegate(y[i],yy[i].input);
        end;

        for i := 0 to len(mul,0)−1 do
            for j := 0 to len(mul,1)−1 do
                new(mul[i,j]);
                delegate(x[i,j], mul[i,j].input [0]);
            connect(yy[j].output,mul[i,j ]. input [1],0);
          end;
        end;

        for i := 0 to len(sum1d,0)−1 do
          new(sum1d[i]);
          for j := 0 to len(mul,1)−1 do
            connect(mul[i,j].output,sum1d[i].x[j ],0);
          end;
          delegate(z[i],sum1d[i].y);
        end;

    end Contract2d1d;

    InnerProd1d = cellnet(
                         x: array Dim1d of port in;
                         y: array Dim1d of port in;
                         z: port out
                         );
    var
        i: longint;
        mul: array Dim1d of MulType;
        sum1d: Sum1d;
    begin
        new(sum1d);
        for i := 0 to len(mul,0)−1 do
          new(mul[i]);
          delegate(x[i],mul[i ]. input [0]);
          delegate(y[i],mul[i ]. input [1]);
          connect(mul[i].output,sum1d.x[i],0);
        end;

        delegate(z,sum1d.y);
    end InnerProd1d;

    (* Contraction of a 3−D tensor with three 1−D tensors *)
    NusiLsContract3d = cellnet(
                           x: array Dim1d of array Dim1d of array Dim1d of port in;
                           u, v, w: array Dim1d of port in;
                           s: port out
                           );
    var
        contract3d1d: Contract3d1d;
        contract2d1d: Contract2d1d;
        innerProd1d: InnerProd1d;
        i, j, k: longint;
    begin
        new(contract3d1d);
        new(contract2d1d);
        new(innerProd1d);

        for k := 0 to len(w,0)−1 do delegate(w[k],contract3d1d.y[k]); end;
        for k := 0 to len(v,0)−1 do delegate(v[k],contract2d1d.y[k]); end;
        for k := 0 to len(u,0)−1 do delegate(u[k],innerProd1d.y[k]); end;

        for i := 0 to len(x,0)−1 do
          for j := 0 to len(x,1)−1 do
            for k := 0 to len(x,2)−1 do
              delegate(x[i,j ,k ], contract3d1d.x[i , j ,k ]);
            end;
            connect(contract3d1d.z[i,j ], contract2d1d.x[i , j ],0);
          end;
          connect(contract2d1d.z[i],innerProd1d.x[i ],0);
        end;
    delegate(s,innerProd1d.z);
  end NusiLsContract3d;

  (* Full tensor computational kernel used for computing LS part of the gradient
     in tensor B−spline−based signal reconstruction algorithm
  *)
  NusiLsKernel3d = cellnet(
                          x: array Dim1d of array Dim1d of array Dim1d of port in;
                          u, v, w: array Dim1d of port in;
                          y0: array Dim1d of array Dim1d of array Dim1d of port in;
                          y: array Dim1d of array Dim1d of array Dim1d of port out
                          );
  var
    i, j, k: longint;
    nusiLsContract3d: NusiLsContract3d;
    tensorProduct3d: TensorProduct3d;
    mul: array Dim1d of MulType;
    add: array Dim1d of array Dim1d of array Dim1d of AddType;
  begin
    new(nusiLsContract3d);
    new(tensorProduct3d);

    for k := 0 to len(u,0)−1 do
      delegate(u[k],nusiLsContract3d.u[k]);
      delegate(v[k],nusiLsContract3d.v[k]);
      delegate(w[k],nusiLsContract3d.w[k]);
    end;

    for i := 0 to len(x,0)−1 do
      for j := 0 to len(x,1)−1 do
        for k := 0 to len(x,2)−1 do
          delegate(x[i,j,k], nusiLsContract3d.x[i,j,k]);
        end;
      end;
    end;

    (* scale one of the vectors with the contraction result *)
    for k := 0 to len(u,0)−1 do
      new(mul[k]);
      connect(nusiLsContract3d.s,mul[k].input[0],0);
      delegate(u[k],mul[k].input [1]);
    end;

    (* compute the tensor product *)
    for k := 0 to len(u,0)−1 do connect(mul[k].output,tensorProduct3d.x[k],0); end;
    for k := 0 to len(v,0)−1 do delegate(v[k],tensorProduct3d.y[k]); end;
    for k := 0 to len(w,0)−1 do delegate(w[k],tensorProduct3d.z[k]); end;

    (* increment *)
    for i := 0 to len(y,0)−1 do
      for j := 0 to len(y,1)−1 do
        for k := 0 to len(y,2)−1 do
          new(add[i,j,k]);
          delegate(y0[i,j,k], add[i,j,k].input [0]);
          connect(tensorProduct3d.xyz[i,j,k],add[i,j,k].input [1],0);
          delegate(y[i,j,k], add[i,j,k].output);
        end
      end
    end;
  end NusiLsKernel3d;

end NusiFpga.
```

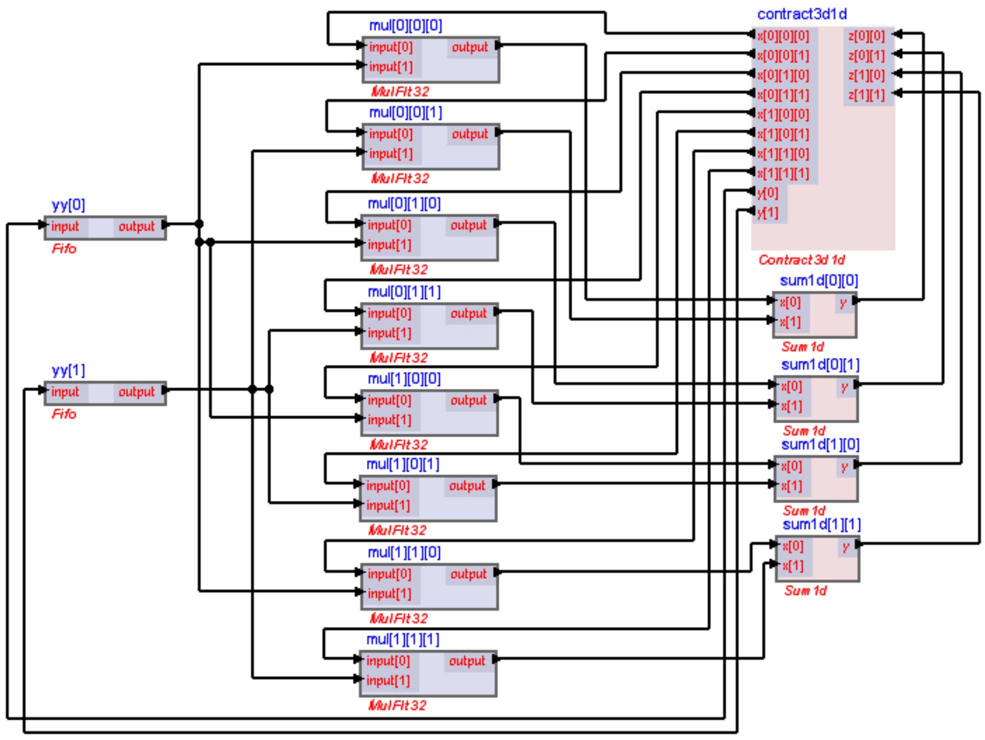


Fig. 5.6: A block diagram of Contract3d1d cellnet that corresponds to the product $\mathcal{H}_z \cdot \mathcal{C}^{xyz}$ from the equation (5.2) for B-spline degree $n = 1$

5.5 Discussion

We introduced a high-level framework for the design of high-performance streaming signal processing systems on FPGA. The proposed approach allows to build complete industrial quality FPGA hardware/software systems in a short time and is easily comprehensible to software and signal processing engineers who are not experts in the field of FPGA design. The proposed approach combines the flexibility of multi-processor systems and high performance of dedicated application specific circuits. These features make it particularly well suitable for solving the signal reconstruction problem considered in this work. We showed how the developed framework can be efficiently used for implementation of our tensor- and spline-based algorithms.

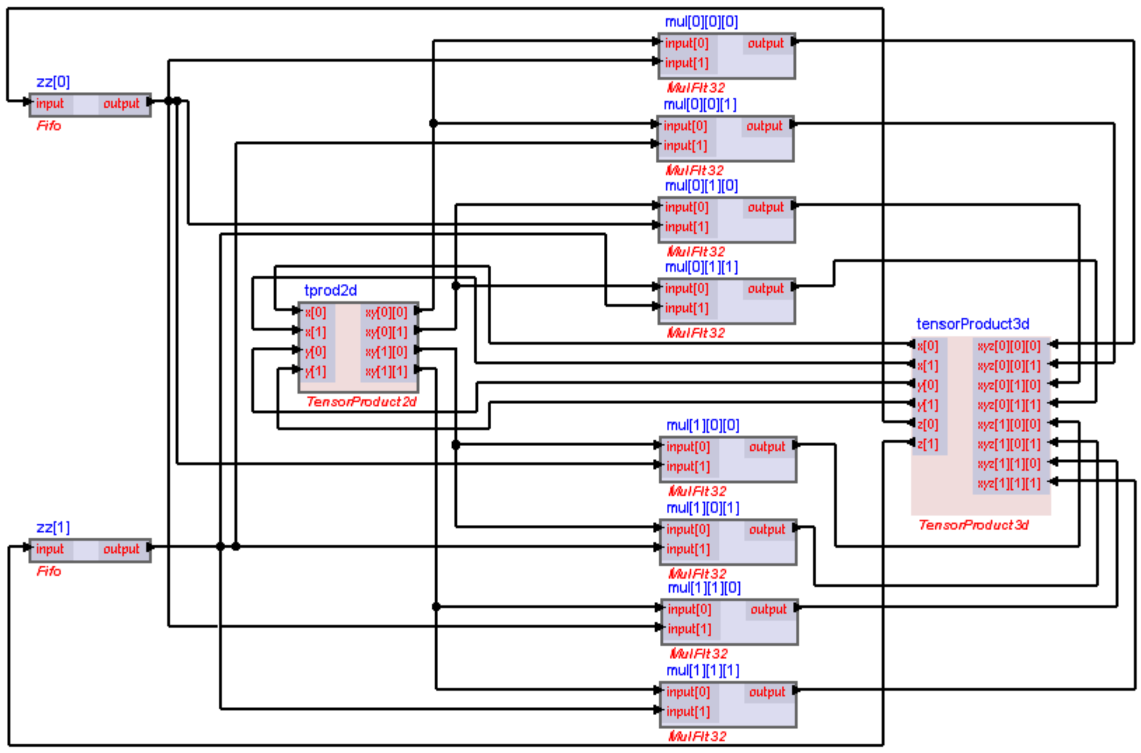


Fig. 5.7: A block diagram of TensorProduct3d cellnet that implements tensor product of three one-dimensional tensors

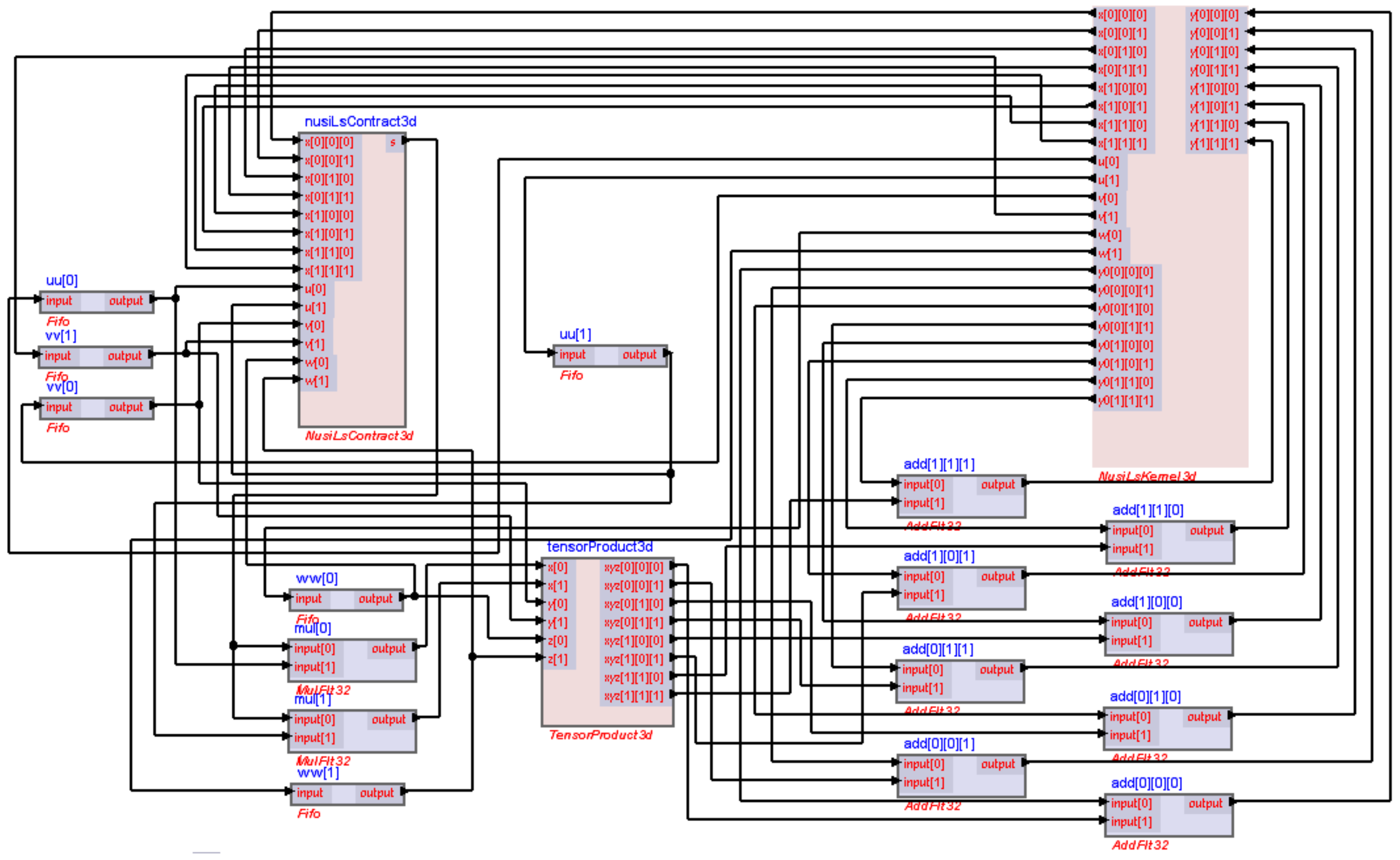


Fig. 5.8: A block diagram of the whole computational kernel implementing the equation (5.2)

Chapter 6

Summary

In this work we performed a detailed structural analysis of the computational problem of spline-based variational signal reconstruction. In the first part of the work we introduced a tensor-based abstraction that allowed us to formulate the considered problem in multiple dimensions with preservation of the underlying multidimensional computational structure. This differentiating feature of the proposed abstraction facilitated the development of a computationally efficient iterative algorithm for the reconstruction of large multidimensional images from arbitrarily sampled data. The derived algorithm was successfully applied to a real-life medical imaging problem: reconstruction of 4-D (3-D+time) echocardiographic signal from a very large set of non-uniform measurements.

Then we considered another approach to the signal reconstruction problem that is based on a non-iterative computation of variational spline solutions via inverse recursive filtering. As a result we derived highly-efficient algorithms for computing the inverse smoothing spline filters and the corresponding solutions for an arbitrary spline order with the possibility to optimize the solution using a statistical optimality measure that is based on generalized cross-validation. We introduced an extension of the classical smoothing spline regularization that allows to compute smoothing spline filters with improved noise discrimination characteristics. We extended our results obtained for uniform smoothing splines and derived high performance algorithms for one-dimensional non-uniform signal reconstruction. We successfully applied the proposed algorithms to the problem of medical Optical Coherence Tomography. Finally, we introduced an extension of the proposed algorithms to the case of multidimensional signal reconstruction. We did not study the proposed multidimensional extension in full detail and left this problem for consideration in our future research.

In the last part of this work we introduced a high-level approach to design of high-performance data processing systems on FPGA. The proposed approach combines the flexibility of multi-processor systems and the high performance of dedicated application specific circuits. It is easily comprehensible to software and signal processing engineers who are not FPGA experts and, therefore, opens the door to innovative high-level embedded system designs. We showed that the presented framework is particularly well-suitable for solving the signal reconstruction problem considered in this work.

Bibliography

[1] C. De Boor, *A practical guide to splines.* New York: Springer, 1978.

[2] A. Aldroubi and K. Gröchenig, “Nonuniform sampling and reconstruction in shift-invariant spaces,” *SIAM Rev.*, vol. 43, no. 4, pp. 585–620, Apr. 2001. [Online]. Available: http://dx.doi.org/10.1137/S0036144501386986

[3] M. Unser and T. Blu, “Generalized smoothing splines and the optimal discretization of the Wiener filter,” *IEEE Transactions on Signal Processing*, vol. 53, no. 6, pp. 2146–2159, June 2005.

[4] M. Arigovindan, M. Suehling, P. Hunziker, and M. Unser, “Variational Image Reconstruction from Arbitrarily Spaced Samples: A Fast Multiresolution Spline Solution,” *IEEE Transactions on Image Processing*, vol. 14, no. 4, pp. 450–460, 2005.

[5] M. Unser, “Splines: A perfect fit for signal and image processing,” *IEEE Signal Processing Magazine*, vol. 16, no. 6, pp. 22–38, November 1999, iEEE Signal Processing Society’s 2000 magazine award.

[6] M. Unser, A. Aldroubi, and M. Eden, “B-Spline signal processing: Part I—Theory,” *IEEE Transactions on Signal Processing*, vol. 41, no. 2, pp. 821–833, February 1993, IEEE Signal Processing Society’s 1995 best paper award. [Online]. Available: http://bigwww.epfl.ch/publications/unser9301.html

[7] ——, “B-Spline signal processing: Part II—Efficient design and applications,” *IEEE Transactions on Signal Processing*, vol. 41, no. 2, pp. 834–848, February 1993. [Online]. Available: http://bigwww.epfl.ch/publications/unser9302.html

[8] L. Elden, “Multi-linear mappings, SVD, HOSVD, and the numerical solution of ill-conditioned tensor least squares problems,” in *Workshop on Tensor Decompositions and Applications TDA05*, 2005.

[9] V. Pereyra and G. Scherer, “Efficient Computer Manipulation of Tensor Products with Applications to Multidimensional Approximation,” *Mathematics of Computation*, vol. 27, no. 123, pp. 595–605, 1973.

[10] G. Baumgartner, E. Auer, D. E. Bernholdt, A. Bibireata, D. Cociorva, X. Gao, S. Krishnan, R. J. Harrison, C. chung Lam, Q. Lu, and M. Nooijen, “Synthesis of

High-Performance Parallel Programs for a Class of Ab Initio Quantum Chemistry Models," in *Proceedings of the IEEE*, 2005, pp. 276–292.

[11] R. E. Lynch, J. R. Rice, and D. H. Thomas, "Tensor product analysis of partial difference equations," *Bull. Amer. Math. Soc.*, vol. 70, 3, pp. 378–384, 1964.

[12] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, September 2009.

[13] D. Muti and S. Bourennane, "Multidimensional signal processing using lower-rank tensor approximation," *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, vol. 3, pp. III–457–60 vol.3, April 2003.

[14] P. Comon, "Tensor decompositions, state of the art and applications," *IMA Conference Mathematics in Signal Processing, Warwick, UK*, 2000.

[15] W. Hongcheng and N. Ahuja, "Compact representation of multidimensional data using tensor rank-one decomposition," *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 1, pp. 44–47 Vol.1, Aug. 2004.

[16] C. M. Martin. (2004) Tensor Decompositions Workshop Discussion Notes.

[17] O. Morozov and P. Hunziker, "Solving tensor structured problems with computational tensor algebra," *CoRR*, vol. abs/1001.5460, 2010.

[18] R. E. Blahut, *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, January 1985.

[19] B. Guoan and Z. Yonghong, *Transforms and fast algorithms for signal analysis and representations.* Boston : Birkhauser, 2004.

[20] A. Aldroubi and M. Unser, *Wavelets in Medicine and Biology.* Boca Raton FL, USA: CRC Press, 1996, 616 p.

[21] L. Zeng, C. Jansen, S. Marsch, M. Unser, and P. Hunziker, "Four-dimensional wavelet compression of arbitrarily sized echocardiographic data," *IEEE Transactions on Medical Imaging*, vol. 21, no. 9, pp. 1179–1187, September 2002.

[22] J. H. Heinbockel, *Introduction to tensor calculus and continuum mechanics.* Trafford Publishing, Norfolk, Virginia, 2001.

[23] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, pp. 279–311, 1966.

[24] D. Leibovici and R. Sabatier, "A Singular Value Decomposition of a k-Way Array for a Principal Component Analysis of Multiway Data, PTA-k," *Linear Algebra and its Applications*, vol. 269, pp. 307–329, 1998.

[25] L. D. Lathauwer, B. D. Moor, and J. Vandewalle, “A Multilinear Singular Value Decomposition,” *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1253–1278, 2000.

[26] R. A. Harshman, “Foundations of the PARAFAC procedure: Model and conditions for an ÂńexplanatoryÂż multi-mode factor analysis,” *UCLA Working Papers in phonetics*, vol. 16, pp. pp. 1–84, 1970.

[27] E. Acar and B. Yener, “Unsupervised Multiway Data analysis: A Literature Survey,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 21, no. 1, pp. 6–20, 2009.

[28] A. Einstein, “The Foundation of the General Theory of Relativity,” in *The Collected Papers of Albert Einstein.* Princeton University Press, 1997, vol. 6, pp. 146—200.

[29] R. Smilde, A. Bro and P. Geladi, *Multi-Way Analysis: Applications in the Chemical Sciences.* Wiley, West Sussex, England, 2004.

[30] O. V. Morozov, M. Unser, and P. R. Hunziker, “Reconstruction of large, irregularly sampled multidimensional images. a tensor-based approach,” *IEEE Trans. Med. Imaging*, vol. 30, no. 2, pp. 366–374, 2011.

[31] L. D. Lathauwer and B. D. Moor, “From Matrix to Tensor : Multilinear Algebra and Signal Processing,” *Mathematics in Signal Processing IV, McWhirter J., ed.*, pp. 1–15, 1998, selected papers presented at 4th IMA Int. Conf. on Mathematics in Signal Processing.

[32] G. Wahba, *Spline models for observational data.* Society for Industrial and Applied Mathematics, 1990.

[33] M. D. Buhmann, *Radial basis functions : theory and implementations.* Cambridge University Press, 2003.

[34] J. Duchon, *Splines minimizing rotation-invariant semi-norms in Sobolev spaces*, ser. Lecture Notes in Mathematics. Springer Berlin / Heidelberg, 1977, vol. 571.

[35] F. L. Bookstein, “Principal warps: thin-plate splines and the decomposition of deformations,” vol. 11, no. 6, pp. 567–585, Jun. 1989.

[36] R. K. Beatson, J. B. Cherrie, and D. L. Ragozin, “Fast evaluation of radial basis functions: Methods for four-dimensional polyharmonic splines,” *SIAM Journal on Mathematical Analysis*, vol. 32, no. 6, pp. 1272–1310, 2001.

[37] D. Achlioptas, F. Mcsherry, and B. Schoelkopf, “Sampling Techniques for Kernel Methods,” pp. 335–342, 2001.

[38] H. Zhang and M. Genton. (2008) Compactly supported Radial Basis Function Kernels. CiteSeerX.

[39] E. Larsson and B. Fornberg, “Theoretical and computational aspects of multivariate interpolation with increasingly flat radial basis functions,” *Comput. Math. Appl*, vol. 49, pp. 103–130, 2003.

[40] R. K. Beatson, W. A. Light, and S. Billings, “Fast solution of the radial basis function interpolation equations: Domain decomposition methods,” *SIAM J. Sci. Comput.*, vol. 22, no. 5, pp. 1717–1740, 2000.

[41] M. Bertram, J. C. Barnes, B. Hamann, K. I. Joy, H. Pottmann, and D. Wushour, “Piecewise optimal triangulation for the approximation of scattered data in the plane,” *Comput. Aided Geom. Des.*, vol. 17, no. 8, pp. 767–787, 2000.

[42] Y. Jang, R. P. Botchen, A. Lauser, D. S. Ebert, K. P. Gaither, and T. Ertl, “Enhancing the interactive visualization of procedurally encoded multifield data with ellipsoidal basis functions.” *Comput. Graph. Forum*, vol. 25, no. 3, pp. 587–596, 2006.

[43] D. Juba and A. Varshney, “Modelling and rendering large volume data with gaussian radial basis functions,” *Tech. Rep., University of Maryland*, 2007.

[44] M. Arigovindan, M. Sühling, P. Hunziker, and M. Unser, “Multigrid image reconstruction from arbitrarily spaced samples,” in *Proceedings of the 2002 IEEE International Conference on Image Processing (ICIP’02)*, vol. III, Rochester NY, USA, September 22-25, 2002, pp. 381–384.

[45] E. Vucini, T. Moeller, and M. E. Groeller, “Efficient reconstruction from non-uniform point sets,” *The Visual Computer*, 2008.

[46] E. Vuçini, T. Möller, and M. E. Gröller, “On visualization and reconstruction from non-uniform point sets using b-splines,” *Computer Graphics Forum*, vol. 28, no. 3, pp. 1007–1014, Jun. 2009, 2nd Best Paper Award.

[47] F. Friedrich and J. Gutknecht, “Array-Structured Object Types for Mathematical Programming,” *Lecture Notes in Computer Science*, vol. 4228, pp. pp. 195–210, 2006.

[48] M. Siebenthal, G. Szekely, U. Gamper, P. Boesiger, A. Lomax, and P. Cattin, “4D MR imaging of respiratory organ motion and its variability,” *Physics in Medicine and Biology*, vol. 52, pp. 1547–64, 2007.

[49] K. Djoa and N. de Jong, “A fast rotating scanning unit for real-time three-dimensional echo data acquisition,” *Ultrasound Med Biol*, vol. 26(5), pp. pp. 863–9, 2000.

[50] M. Voormolen and B. Krenning, “A New Transducer for 3d Harmonic imaging,” *Proceedings of the IEEE Ultrasonics Symposium*, pp. pp. 1261–1264, 2002.

[51] Y. Saad and H. A. van der Vorst, “Iterative solution of linear systems in the 20th century,” *J. Comput. Appl. Math.*, vol. 123, no. 1-2, pp. 1–33, 2000.

[52] I. J. Schoenberg, "Spline functions and the problem of graduation," *Proc. Nat. Acad. Sci.*, vol. 52, pp. 947–950, 1964.

[53] C. REINSCH, "Smoothing by spline functions." *Numerische Mathematik*, vol. 10, pp. 177–183, 1967. [Online]. Available: http://eudml.org/doc/131782

[54] M. F. Hutchinson and F. R. de Hoog, "Smoothing noisy data with spline functions," *Journal of Numerical Mathematics*, vol. 47, no. 1, pp. 99–106, Aug. 1985.

[55] F. MacWilliams, "An iterative method for the direct hurwitz-factorization of a polynomial," *Circuit Theory, IRE Transactions on*, vol. 5, no. 4, pp. 347 – 352, 1958.

[56] G. Wilson, "Factorization of the covariance generating function of a pure moving average process," *SIAM Journal on Numerical Analysis*, vol. 6, no. 1, pp. 1–7, 1969. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/0706001

[57] F. L. Bauer, "Ein direktes Iterationsverfahren zur Hurwitz-Zerlegung eines Polynoms," *Arch. Elek. Übertr.*, vol. 9, pp. 285–290, 1955.

[58] S. Fomel, P. Sava, J. Rickett, and J. F. Claerbout, "The wilson-burg method of spectral factorization with application to helical filtering," *Geophysical Prospecting*, vol. 51, no. 5, pp. 409–420, 2003. [Online]. Available: http://dx.doi.org/10.1046/j.1365-2478.2003.00382.x

[59] K. Sidek, F. Sufi, I. Khalil, and D. Al-Shammary, "An efficient method of biometric matching using interpolated ecg data," in *Biomedical Engineering and Sciences (IECBES), 2010 IEEE EMBS Conference on*, Nov 2010, pp. 330–335.

[60] P. Craven and G. Wahba, "Smoothing noisy data with spline functions," *Numerische Mathematik*, vol. 31, no. 4, pp. 377–403, 1978. [Online]. Available: http://dx.doi.org/10.1007/BF01404567

[61] R. E. O. Dursun Aydin, Memmedaga Memmedli, "Smoothing parameter selection for nonparametric regression using smoothing spline," *EUROPEAN JOURNAL OF PURE AND APPLIED MATHEMATICS*, vol. 6, pp. 222–238, 2013.

[62] H. L. Weinert, *Fast Compact Algorithms and Software for Spline Smoothing*, ser. Springer Briefs in Computer Science. Springer, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-5496-0

[63] F. Friedrich, J. Gutknecht, O. Morozov, and P. Hunziker, "Mathematical programming language extension for mathematical programming language extension for multilinear algebra," *Proc. Kolloqium über Programmiersprachen und Grundlagen der Programmierung*, 2007.

[64] J. Dongarra, "Basic Linear Algebra Subprograms Technical Forum Standard," *International Journal of High Performance Applications and Supercomputing*, vol. 16, no. 1, pp. 1–111, 2002.

[65] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate," *ACM Transactions on Mathematical Software*, vol. 35, no. 3, pp. 22:1–22:14, Oct. 2008. [Online]. Available: http://doi.acm.org/10.1145/1391989.1391995

[66] N. I. M. Gould, J. A. Scott, and Y. Hu, "A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations," *ACM Trans. Math. Softw.*, vol. 33, no. 2, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1236463.1236465

[67] J. G. Fujimoto, C. Pitris, S. A. Boppart, and M. E. Brezinski, "Optical coherence tomography: An emerging technology for biomedical imaging and optical biopsy," *Neoplasia*, vol. 2, no. 1–2, pp. 9 – 25, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1476558600800172

[68] A. M. Zysk, F. T. Nguyen, A. L. Oldenburg, D. L. Marks, and S. A. Boppart, "Optical coherence tomography: a review of clinical development from bench to bedside," *Journal of Biomedical Optics*, vol. 12, no. 5, pp. 051 403–051 403–21, 2007. [Online]. Available: http://dx.doi.org/10.1117/1.2793736

[69] H. G. Bezerra, M. A. Costa, G. Guagliumi, A. M. Rollins, and D. I. Simon, "Intracoronary optical coherence tomography: A comprehensive review: Clinical and research applications," *JACC: Cardiovascular Interventions*, vol. 2, no. 11, pp. 1035 – 1046, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1936879809005925

[70] S. Vergnole, D. Lévesque, and G. Lamouche, "Experimental validation of an optimized signal processing method to handle non-linearity in swept-source optical coherence tomography," vol. 18, no. 10, pp. 10 446–10 461+, 2010. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-18-10-10446

[71] J. A. Fessler and B. P. Sutton, "Nonuniform fast fourier transforms using min-max interpolation." *IEEE Transactions on Signal Processing*, vol. 51, no. 2, pp. 560–574, 2003. [Online]. Available: http://dblp.uni-trier.de/db/journals/tsp/tsp51.html#FesslerS03

[72] T. S. Chan KK, "Selection of convolution kernel in non-uniform fast fourier transform for fourier domain optical coherence tomography," *Opt Express*, 2011.

[73] J. Izatt and M. Choma, "Theory of optical coherence tomography," in *Optical Coherence Tomography*, ser. Biological and Medical Physics, Biomedical Engineering, W. Drexler and J. Fujimoto, Eds. Springer Berlin Heidelberg, 2008, pp. 47–72. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-77550-8_2

[74] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[75] L. Liu and O. Morozov, "A process-oriented streaming system design paradigm for fpgas." in *ReConFig*, V. K. Prasanna, J. Becker, and R. Cumplido, Eds. IEEE Computer Society, 2010, pp. 370–375. [Online]. Available: http://dblp.uni-trier.de/db/conf/reconfig/reconfig2010.html#LiuM10

[76] L. Liu, O. Morozov, Y. Han, J. Gutknecht, and P. R. Hunziker, "Automatic soc design flow on many-core processors: a software hardware co-design approach for fpgas." in *FPGA*, J. Wawrzynek and K. Compton, Eds. ACM, 2011, pp. 37–40. [Online]. Available: http://dblp.uni-trier.de/db/conf/fpga/fpga2011.html#LiuMHGH11

[77] N. Wirth, "The tiny register machine (trm)," 2009. [Online]. Available: http://e-collection.library.ethz.ch/eserv/eth:4969/eth-4969-01.pdf

[78] F. Friedrich, L. Liu, and J. Gutknecht, "Active cells: A computing model for rapid construction of on-chip multi-core systems." in *ACIS-ICIS*, H. Miao, R. Y. Lee, H. Zeng, and J. Baik, Eds. IEEE, 2012, pp. 463–469. [Online]. Available: http://dblp.uni-trier.de/db/conf/ACISicis/ACISicis2012.html#FriedrichLG12

[79] ARM, "Amba 4 axi4-stream protocol, specification version 1.0," 2010.

[80] S. Sharma and W. Chen, "Using model-based design to accelerate fpga development for automotive applications," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.*, pp. 150–158, 2009.

[81] ARM, "Arm amba axi protocol v2.0 specification."

[82] T. Kaegi-Trachsel and J. Gutknecht, "Minos - the design and implementation of an embedded real-time operating system with a perspective of fault tolerance." in *IMCSIT*. IEEE, 2008, pp. 649–656. [Online]. Available: http://dblp.uni-trier.de/db/conf/imcsit/imcsit2008.html#Kaegi-TrachselG08

[83] P. Muller, "The active object system design and multiprocessor implementation," Ph.D. dissertation, ETHZ, 2002. [Online]. Available: http://e-collection.library.ethz.ch/eserv/eth:26082/eth-26082-02.pdf