# Algorithmic Differentiation: Application to Variational Problems in Computer Vision

Thomas Pock, Michael Pock, and Horst Bischof

***Abstract*— Many vision problems can be formulated as minimization of appropriate energy functionals. These energy functionals are usually minimized, based on the calculus of variations (Euler-Lagrange equation). Once the Euler-Lagrange equation has been determined it needs to be discretized in order to implement it on a digital computer. This is not a trivial task and, moreover, error-prone. In this article, we propose a flexible alternative. We discretize the energy functional and subsequently apply the mathematical concept of algorithmic differentiation to directly derive algorithms that implement the energy functional's derivatives. This approach has several advantages: First, the computed derivatives are exact with respect to the implementation of the energy functional. Second, it is basically straightforward to compute second-order derivatives and thus the Hessian matrix of the energy functional. Third, algorithmic differentiation is a process which can be automated. We demonstrate this novel approach on three representative vision problems (namely denoising, segmentation, and stereo) and show that state-of-the-art results are obtained with little effort.**

***Index Terms*— Evaluating derivatives, algorithmic differentiation, variational methods, energy functional, optimization**

## I. Introduction

COMPUTER vision inherently deals with the problem of determining unknown quantities, based on the observation of their effects (e. g. recovering the 3D structure of a stereo image pair). Such problems are referred to as "inverse problems" and are typically ill-posed in the original sense of Hadamard [1]. Examples of inverse problems arising in computer vision can be found, for example, in [2], [3], and [4]. Since the beginning of computer vision, a great number of methods have been developed to well-pose (i. e. regularize) inverse vision problems, and thus to facilitate the computation of the unknown quantities. Independent of the application, the major goal of image modelling is to find proper ways to mathematically describe and analyze images.

An inverse problem can often be solved by minimization of an energy functional which appropriately describes the behaviour of the associated model. Mathematically speaking, we consider the following class of minimization problems:

$$\mathbf{x}^* = \inf_{\mathbf{x}} \left\{ y = f\left(\mathbf{x}\right) \right\}, \tag{1}$$

where $y = f(\mathbf{x})$ is the energy functional, $\mathbf{x}$ is the set of input parameters to be optimized, and $\mathbf{x}^*$ denotes the optimal solution. In computer vision, energy functionals are typically of the form

$$y = f_{\text{Data}}\left(\mathbf{u}, \mathbf{g}\right) + \alpha\, f_{\text{Reg}}\left(\mathbf{u}\right), \tag{2}$$

with $f_{\text{Data}}(\mathbf{u}, \mathbf{g})$ a data term measuring the energy of the solution $\mathbf{u}$ with respect to the observed data $\mathbf{g}$ and $f_{\text{Reg}}(\mathbf{u})$ a regularization term measuring the smoothness of the solution itself. The free parameter $\alpha$ is used to control the amount of regularization. Various energy functionals of this type can be found for quite fundamental vision tasks, e. g. optical flow computation [5] and segmentation [6].

The calculus of variations includes a framework for finding the solution of such minimization problems. The fundamental theorem of this approach, the Euler-Lagrange differential equation, provides a *necessary* condition to describe a functional at stationary points. It can therefore be used to determine those parameters for which the functional yields a maximum or minimum value. In the course of the optimization process, derivatives of the energy functional with respect to the model parameters have to be evaluated. However, the derivation of the Euler-Lagrange equations can be tedious if the energy functional is complex. Moreover (since we are dealing with discrete images), the discretization and efficient numerical implementation of partial differential equations such as Euler-Lagrange equations are a pending issue and, *per se*, a scientific field [7].

In this article, we propose a flexible alternative to the analytical derivation of the Euler-Lagrange equations, based on the mathematical concept of algorithmic differentiation. Algorithmic differentiation is well-known in meteorology [8], but has, to the best of our knowledge, not been used in computer vision so far. In contrast to the Euler-Lagrange formalism, derivatives of the energy functional are obtained by directly differentiating the energy functional as implemented in an appropriate programming language (e. g. Fortran or C/C++). This approach has several advantages:

First, it is *self-contained*. This implies that the computed derivatives are exact with respect to the algorithmic representation of the energy functional [9]. Furthermore, if the energy functional is implemented using more accurate schemes the implementation of the derivatives yields more accurate results as well [10]. Second, it is basically straightforward to compute second-order derivatives and thus the Hessian matrix of the energy functional. With this, the energy functional can be minimized by means of Newton-type optimization methods. Third, algorithmic differentiation can be done automatically using automatic differentiation (AD) tools [11]. Especially for complex models, this opens a lot of new perspectives to the design and study of computer vision methods.

Thomas Pock and Horst Bischof are with the Institute for Computer Graphics and Vision, Graz University of Technology, Inffeldgasse 16, A-8010 Graz, Austria. E-mail: {pock, bischof}@icg.tugraz.at

Michael Pock is with the Wegener Center for Climate and Global Change, University of Graz, Leechgasse 25, A-8010 Graz, Austria. E-mail: michael.pock@uni-graz.at

The remainder of this article is organized as follows: In section II, we provide a concise introduction to the basic mathematics of algorithmic differentiation. In section III, we illustrate, in detail, the practical application of algorithmic differentiation by considering a simple image regularization model. In section IV, we apply algorithmic differentiation to three common vision problems, namely denoising, segmentation, and stereo. Finally, in section V, we give some conclusions and suggest possible directions for future investigation.

## II. Algorithmic Differentiation

Algorithmic differentiation is a mathematical concept whose relevance to natural sciences has steadily been increasing in the last twenty years. This proposition is especially true of meteorology where, for a long time, algorithmic differentiation has played a key role in adjoint modelling and data assimilation [8], [12], [13]. Since differentiating algorithms is, in principle, tantamount to applying the chain rule of differential calculus [14] the theoretic fundamentals of algorithmic differentiation are long-established. However, only recent progress in the field of computer science places us in a position to widely exploit its capabilities [10].

In literature, we frequently find algorithmic differentiation to be referred to as "automatic differentiation" [9]. The latter term is preferably used by mathematicians and computer scientists who are concerned with designing software implementations of algorithmic differentiation, called "automatic differentiation (AD) tools" [11]. Roughly speaking, there are two elementary approaches to accomplishing this rather challenging task, namely source transformation and operator overloading. Prominent examples of AD tools performing source transformation include "Automatic Differentiation of Fortran" (ADIFOR) [15], "Transformation of Algorithms in Fortran" (TAF) [16], and "Tapenade" [17]. The best-known AD tool implementing the operator overloading approach is "Automatic Differentiation by Overloading in C++" (ADOL-C) [18]. Finally, since it is the AD tool we have actually tested, we mention "Transformation of Algorithms in C++" (TAC++) [19]. Being still in its planning phase, TAC++ has not arrived at full functionality as yet. Nevertheless, we have successfully applied TAC++ to the total variation denoising model described in section IV.

Principally inspired by [12] and [20], we devote the next four subsections to briefly discussing the basic mathematics of algorithmic differentiation. More precisely, we develop a compact mathematical representation, right from the start focusing on just those aspects that are relevant to the class of computer vision problems consisting in the minimization of a scalar energy functional $y = f(\mathbf{x})$. We demonstrate that any algorithm properly implementing $f$ can be differentiated to generate three derivative algorithms, namely the tangent-linear, adjoint, and Hessian algorithms. In addition, we prove that these three derivative algorithms are particularly suited to the computation of the directional derivative $\partial y/\partial s = \partial y/\|\partial \mathbf{x}\|_2$, the gradient $\partial y/\partial \mathbf{x}$, and the Hessian matrix-vector product $\left(\partial^2 y/\partial \mathbf{x}^2\right) \cdot \mathbf{s}$, respectively.

### A. Definition of an Algorithm

Let us consider a scalar function $f$ which maps an open subset $X$ of the $P$-dimensional Euclidian space $\mathbb{R}^P$ to a subset $Y$ of the real line $\mathbb{R}$:

$$f:\ X \subseteq \mathbb{R}^P \mapsto Y \subseteq \mathbb{R}. \tag{3}$$

If an element of $X$ is referred to as the column vector $\mathbf{x} = [x_p]_{p=1}^P$ and if $y$ denotes the image of $\mathbf{x}$ in $Y$ we may alternatively write

$$y = f(\mathbf{x}). \tag{4}$$

We assume that, at any point $\mathbf{x} \in X$, $f$ is continuous and has continuous partial derivatives up to the second order. Correspondingly, we regard $f$ as a $C^2$-function on the domain $X$:

$$f \in C^2(X). \tag{5}$$

Clearly, in case of scalar minimization problems as arising in the field of computer vision, many objective functions of practical interest are not as smooth as required by (5). We may, however, expect that any such function satisfies (5) *domainwise*, at least. For explicit information in this regard, see e. g. [10], where a whole chapter is devoted to the challenge of dealing with non-differentiability in the context of algorithmic differentiation.

Now, suppose we have designed an algorithm that properly implements the function $f$. To simplify matters, we here understand by the term "algorithm" a sequence of *mathematical* statements based on the fundamental arithmetic operations and the small set of elementary functions intrinsic to the syntax of a standard programming language such as C/C++ and Fortran. Obviously, when it comes to the practical application of algorithmic differentiation we also need to know how to correctly handle more advanced programming language elements (i. e. loops, conditional statements, procedure calls, input/output statements, pointer assignments,…). For details concerning this issue, see e. g. [20].

In order that it constitutes an implementation of the function $f$, we assume the algorithm to declare a total of $Q \geq P + 1$ scalar variables collected in the column vector $\mathbf{v} = [v_q]_{q=1}^Q$. $Q - P - 1 \geq 0$ of these variables are auxiliary variables used by the algorithm for reasons such as convenience and efficiency. Moreover, let the non-linear operator $\mathbf{A}$ symbolize the totality of statements to be executed. Then, the algorithm may be defined by

$$\begin{gathered}\mathbf{A}:\ V^{(\mathrm{ini})} \subseteq \mathbb{R}^Q \mapsto V^{(\mathrm{fin})} \subseteq \mathbb{R}^Q \\ \mathbf{v}^{(\mathbf{fin})} = \mathbf{A}\left(\mathbf{v}^{(\mathbf{ini})}\right),\end{gathered} \tag{6}$$

where $\mathbf{v}^{(\mathbf{ini})}$ and $\mathbf{v}^{(\mathbf{fin})}$ denote the initial and final values of $\mathbf{v}$, respectively. To keep consistency, we require that every component $A_q$ of $\mathbf{A}$ behaves analogously to $f$ in terms of continuity and differentiability, i. e.

$$A_q \in C^2\left(V^{(\mathrm{ini})}\right) \tag{7}$$

for $q = 1, \ldots, Q$. Without loss of generality, we may furthermore determine the $p$-th variable of the algorithm to initially hold the function argument $x_p$ and the $Q$-th variable to finally provide the function value $y$. Hence, we record:

$$\begin{aligned} \mathbf{x} &= \left[v_p^{(\text{ini})}\right]_{p=1}^{P} \\ y &= v_Q^{(\text{fin})}. \end{aligned} \tag{8}$$

To complete this subsection, it remains to express the algorithm's defining equation (6) in a more detailed form. We start out from the fact that the algorithm can be subdivided into $N$ sequential blocks of statements, each individual block redefining any variable once at most as well as referencing only values which originate from preceding blocks and/or the variable initialization. Owing to the latter specification, the $n$-th block of the algorithm may be captured by

$$\begin{gathered} \mathbf{A^{(n)}} : \; V^{(n-1)} \subseteq \mathbb{R}^Q \mapsto V^{(n)} \subseteq \mathbb{R}^Q \\ \mathbf{v^{(n)}} = \mathbf{A^{(n)}} \left(\mathbf{v^{(n-1)}}\right), \end{gathered} \tag{9}$$

where $n = 1, \ldots, N$. If we subject (9) to a recursive evaluation starting off with $n = N$ and consider the evident definitions

$$\begin{aligned} \mathbf{v^{(0)}} &\equiv \mathbf{v^{(ini)}} \\ \mathbf{v^{(N)}} &\equiv \mathbf{v^{(fin)}} \end{aligned} \tag{10}$$

we eventually obtain by analogy with (6):

$$\mathbf{v^{(fin)}} = \mathbf{A^{(N)}} \circ \mathbf{A^{(N-1)}} \circ \; \cdots \; \circ \mathbf{A^{(1)}} \left(\mathbf{v^{(ini)}}\right). \tag{11}$$

Thus, viewing the algorithm as a sequence of $N$ blocks of statements implicates that the global operator $\mathbf{A}$ has to be understood as a composition of $N$ local operators $\mathbf{A^{(n)}}$.

### B. Tangent-Linear Algorithm

Let us introduce a differential operator $\mathfrak{d}$ which involves merely first-order differential operations. For instance, $\mathfrak{d}$ maps a function to its total differential, its directional derivative, or its gradient. The application of $\mathfrak{d}$ to the basic algorithm's defining equation (6) leads to

$$\mathfrak{d}\mathbf{v^{(fin)}} = \mathbf{J}\left(\mathbf{v^{(ini)}}\right) \cdot \mathfrak{d}\mathbf{v^{(ini)}}, \tag{12}$$

where $\mathbf{J}$ stands for the Jacobian matrix associated with the global operator $\mathbf{A}$:

$$\mathbf{J} = \left[\frac{\partial A_i}{\partial v_j^{(\text{ini})}}\right]_{i,j=1}^{Q}. \tag{13}$$

It is important to point out that $\mathbf{J}$, unlike $\mathbf{A}$, preserves linear combinations. Consequently, $\mathbf{J}$ may be treated as a linear operator.

The algorithm defined by (12) is generally denominated "tangent-linear algorithm" [21]. In order to gain more detailed information thereon, we apply $\mathfrak{d}$ to (9) as well. Thus, we find that the $n$-th block of the tangent-linear algorithm is represented by

$$\mathfrak{d}\mathbf{v^{(n)}} = \mathbf{J^{(n)}}\left(\mathbf{v^{(n-1)}}\right) \cdot \mathfrak{d}\mathbf{v^{(n-1)}}, \tag{14}$$

with $n = 1, \ldots, N$ and $\mathbf{J^{(n)}}$ denoting the Jacobian matrix related to the local operator $\mathbf{A^{(n)}}$:

$$\mathbf{J^{(n)}} = \left[\frac{\partial A_i^{(n)}}{\partial v_j^{(n-1)}}\right]_{i,j=1}^{Q}. \tag{15}$$

It suggests itself to evaluate (14) by means of recursion, just taking up the approach pursued in section II-A. If we do so and recall the definitions presented in (10) we can transform (14) to eventually give

$$\mathfrak{d}\mathbf{v^{(fin)}} = \left[\bigodot_{n=N}^{1} \mathbf{J^{(n)}}\left(\mathbf{v^{(n-1)}}\right)\right] \cdot \mathfrak{d}\mathbf{v^{(ini)}}. \tag{16}$$

The comparison of (16) with (12) exhibits that the global Jacobian matrix $\mathbf{J}$ has to be interpreted as an ordinary product of $N$ local Jacobian matrices $\mathbf{J^{(n)}}$, i. e.

$$\mathbf{J} = \bigodot_{n=N}^{1} \mathbf{J^{(n)}}. \tag{17}$$

Clearly, we could also have arrived at (16) in a direct manner, that is, by letting $\mathfrak{d}$ operate on (11). Hence, the multiple matrix product on the right-hand side of (17) is a straightforward consequence of the well-known chain rule of differential calculus.

Aiming to illustrate the *fundamental* usefulness of the tangent-linear algorithm, we finally concretize the effect of the differential operator $\mathfrak{d}$. More precisely, we examine the case of $\mathfrak{d}$ mapping $\mathbf{v^{(fin)}}$ to its directional derivative, which means setting

$$\mathfrak{d} = \partial / \partial w^{(\text{ini})}, \tag{18}$$

where $\partial w^{(\text{ini})} = \|\partial \mathbf{v^{(ini)}}\|_2$. By insertion of (18) into (12) and with $\hat{\mathbf{w}}^{\mathbf{(ini)}}$ denoting the unit vector codirectional with $\partial \mathbf{v^{(ini)}}$, we have

$$\frac{\partial \mathbf{v^{(fin)}}}{\partial w^{(\text{ini})}} = \mathbf{J}\left(\mathbf{v^{(ini)}}\right) \cdot \hat{\mathbf{w}}^{\mathbf{(ini)}}. \tag{19}$$

As a result, we come to the conclusion that, in order for the tangent-linear algorithm to yield the directional derivative $\partial \mathbf{v^{(fin)}} / \partial w^{(\text{ini})}$, we must simply initialize the tangent-linear variable vector $\mathfrak{d}\mathbf{v}$ with the normalized direction vector $\hat{\mathbf{w}}^{\mathbf{(ini)}}$. Thus, the following equivalence applies:

$$\mathfrak{d}\mathbf{v^{(fin)}} = \frac{\partial \mathbf{v^{(fin)}}}{\partial w^{(\text{ini})}} \;\Leftrightarrow\; \mathfrak{d}\mathbf{v^{(ini)}} = \hat{\mathbf{w}}^{\mathbf{(ini)}}. \tag{20}$$

Inasmuch as the basic algorithm constitutes an implementation of the function $f$ our interest is ultimately confined to the directional derivative $\partial y / \partial s = \partial y / \|\partial \mathbf{x}\|_2$. In any case, if we take (8) into account we see that

$$\frac{\partial y}{\partial s} = \frac{\partial v_Q^{(\text{fin})}}{\partial w^{(\text{ini})}} = \mathfrak{d} v_Q^{(\text{fin})}. \tag{21}$$

So, $\partial y / \partial s$ is just equal to the final value of the $Q$-th tangent-linear variable.

### C. Adjoint Algorithm

We are free to conceive an algorithm which arises in the course of transposing the global Jacobian matrix $\mathbf{J}$. This algorithm is generally known as "adjoint algorithm" [21]. By virtue of (17), we find that the transpose of $\mathbf{J}$ is given by the multiple matrix product

$$\mathbf{J}^T = \bigodot_{n=1}^{N} \mathbf{J}^{(\mathbf{n})^T}. \tag{22}$$

Correspondingly, we note: The transposition of $\mathbf{J}$ involves the inversion of the basic block order. Despite this fact, it is quite conducive to the quantitative description of the adjoint algorithm if we retain the basic block numbering. This leads to our defining the adjoint algorithm by the equation

$$\overline{\mathbf{v}}^{(\mathbf{ini})} = \mathbf{J}^T \left(\mathbf{v}^{(\mathbf{ini})}\right) \cdot \overline{\mathbf{v}}^{(\mathbf{fin})}. \tag{23}$$

So far, we have not provided any argument which motivates the introduction of the adjoint algorithm. On this account, let us check what is the consequence of initializing the adjoint variable vector $\overline{\mathbf{v}}$ with the unit vector parallel to the $v_Q$-axis, that is to say, of setting

$$\overline{\mathbf{v}}^{(\mathbf{fin})} = \hat{\mathbf{v}}_{\mathbf{Q}}. \tag{24}$$

Allowing for this concretization and bearing (13) in mind, we carry out the matrix multiplication on the right-hand side of (23). Thereby, we arrive at the following equivalence:

$$\overline{\mathbf{v}}^{(\mathbf{ini})} = \frac{\partial v_Q^{(\mathrm{fin})}}{\partial \mathbf{v}^{(\mathbf{ini})}} \;\Leftrightarrow\; \overline{\mathbf{v}}^{(\mathbf{fin})} = \hat{\mathbf{v}}_{\mathbf{Q}}. \tag{25}$$

In words: If we initialize the adjoint variable vector $\overline{\mathbf{v}}$ with the basis vector $\hat{\mathbf{v}}_{\mathbf{Q}}$ we induce the adjoint algorithm to compute the gradient $\partial v_Q^{(\mathrm{fin})} / \partial \mathbf{v}^{(\mathbf{ini})}$. Given this fact, there is no doubt about the *fundamental* usefulness of the adjoint algorithm. Indeed, we realize by inclusion of (8) that the final value of $\overline{\mathbf{v}}$ incorporates, *inter alia*, the actually interesting gradient $\partial y / \partial \mathbf{x}$:

$$\frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial v_Q^{(\mathrm{fin})}}{\partial v_p^{(\mathrm{ini})}}\right]_{p=1}^{P} = \left[\overline{v}_p^{(\mathrm{ini})}\right]_{p=1}^{P}. \tag{26}$$

At last, we shall illustrate the procedure for deriving the statements of the adjoint algorithm. To begin with, we insert (22) into (23), thus causing the adjoint algorithm's defining equation to assume the more detailed form

$$\overline{\mathbf{v}}^{(\mathbf{ini})} = \left[\bigodot_{n=1}^{N} \mathbf{J}^{(\mathbf{n})^T} \left(\mathbf{v}^{(\mathbf{n-1})}\right)\right] \cdot \overline{\mathbf{v}}^{(\mathbf{fin})}. \tag{27}$$

In consideration of the definitions specified in (10), we extract from (27) that the $n$-th block of the adjoint algorithm is represented by

$$\overline{\mathbf{v}}^{(\mathbf{n-1})} = \mathbf{J}^{(\mathbf{n})^T} \left(\mathbf{v}^{(\mathbf{n-1})}\right) \cdot \overline{\mathbf{v}}^{(\mathbf{n})}, \tag{28}$$

where $n = 1, \ldots, N$. Now, we return to section II-A to become conscious that (9), and hence (28), might possibly contain a considerable amount of *redundant* information. In point of fact, (9) may be restricted to

$$\begin{gathered}\mathbf{A}_{\mathbf{n}}^{(\mathbf{n})} :\; V_n^{(n-1)} \subseteq \mathbb{R}^{Q_n} \mapsto V_n^{(n)} \subseteq \mathbb{R}^{Q_n} \\ \mathbf{v}_{\mathbf{n}}^{(\mathbf{n})} = \mathbf{A}_{\mathbf{n}}^{(\mathbf{n})} \left(\mathbf{v}_{\mathbf{n}}^{(\mathbf{n-1})}\right),\end{gathered} \tag{29}$$

where $\mathbf{v}_{\mathbf{n}} = [v_{n,q_n}]_{q_n=1}^{Q_n}$ encompasses just those $Q_n \leq Q$ variables which are *de facto* redefined or/and referenced by the $n$-th block of the basic algorithm. Likewise, (28) is reducible to

$$\overline{\mathbf{v}}_{\mathbf{n}}^{(\mathbf{n-1})} = \mathbf{J}_{\mathbf{n}}^{(\mathbf{n})^T} \left(\mathbf{v}_{\mathbf{n}}^{(\mathbf{n-1})}\right) \cdot \overline{\mathbf{v}}_{\mathbf{n}}^{(\mathbf{n})}, \tag{30}$$

with $\mathbf{J}_{\mathbf{n}}^{(\mathbf{n})}$ standing for the Jacobian matrix related to the restricted local operator $\mathbf{A}_{\mathbf{n}}^{(\mathbf{n})}$:

$$\mathbf{J}_{\mathbf{n}}^{(\mathbf{n})} = \left[\frac{\partial A_{n,i_n}^{(n)}}{\partial v_{n,j_n}^{(n-1)}}\right]_{i_n,j_n=1}^{Q_n}. \tag{31}$$

As a result, for the purpose of generating the $n$-th block of the adjoint algorithm, we simply need to write out the equation system implied by (30).

### D. Hessian Algorithm

Suppose we are concerned with a computer vision problem whose solution requires the minimization of the energy functional $y = f(\mathbf{x})$. In this case, it might be advantageous to rely on an optimization method (such as a Hessian-free Newton method [22] or an equivalent line-search method [23]) that is based on products arising from the multiplication of the Hessian matrix $\partial^2 y / \partial \mathbf{x}^{\mathbf{2}}$ by some direction vector $\mathbf{s}$. We demonstrate now that any such products are computable using an algorithm, here referred to as "Hessian algorithm", which results from linearizing the adjoint algorithm.

On this note, let us apply the differential operator $\eth$ from section II-B to the adjoint algorithm's defining equation (23). Thereby, we have

$$\eth\overline{\mathbf{v}}^{(\mathbf{ini})} = \eth\mathbf{J}^T \cdot \overline{\mathbf{v}}^{(\mathbf{fin})} + \mathbf{J}^T \left(\mathbf{v}^{(\mathbf{ini})}\right) \cdot \eth\overline{\mathbf{v}}^{(\mathbf{fin})}. \tag{32}$$

With respect to the $Q \times Q$ matrix $\eth\mathbf{J}^T$, we can prove that

$$\begin{aligned}\eth\mathbf{J}^T = \Big[&\mathbf{H_1}\left(\mathbf{v}^{(\mathbf{ini})}\right) \cdot \eth\mathbf{v}^{(\mathbf{ini})}, \mathbf{H_2}\left(\mathbf{v}^{(\mathbf{ini})}\right) \cdot \eth\mathbf{v}^{(\mathbf{ini})}, \\ &\ldots, \mathbf{H_Q}\left(\mathbf{v}^{(\mathbf{ini})}\right) \cdot \eth\mathbf{v}^{(\mathbf{ini})}\Big],\end{aligned} \tag{33}$$

where $\mathbf{H_q}$ represents the Hessian matrix associated with the $q$-th component of the global operator $\mathbf{A}$, meaning

$$\mathbf{H_q} = \left[\frac{\partial^2 A_q}{\partial v_j^{(\mathrm{ini})} \partial v_i^{(\mathrm{ini})}}\right]_{i,j=1}^{Q} \tag{34}$$

for $q = 1, \ldots, Q$. In view of the latter two relations, it is straightforward to conclude from (32) that the equivalence

$$\eth\overline{\mathbf{v}}^{(\mathbf{ini})} = \frac{\partial^2 v_Q^{(\mathrm{fin})}}{\partial \mathbf{v}^{(\mathbf{ini})\mathbf{2}}} \cdot \mathbf{w}^{(\mathbf{ini})} \;\Leftrightarrow\; \begin{cases} \overline{\mathbf{v}}^{(\mathbf{fin})} &= \hat{\mathbf{v}}_{\mathbf{Q}} \\ \eth\overline{\mathbf{v}}^{(\mathbf{fin})} &= \boldsymbol{\emptyset} \\ \eth\mathbf{v}^{(\mathbf{ini})} &= \mathbf{w}^{(\mathbf{ini})} \end{cases} \tag{35}$$

holds for an arbitrary direction vector $\mathbf{w}^{(\mathbf{ini})}$, provided $\hat{\mathbf{v}}_{\mathbf{Q}}$ denotes the unit vector codirectional with the $v_Q$-axis. Eventually, if we take (8) into consideration and allow for

$$\mathbf{s} = \left[ w_p^{(\text{ini})} \right]_{p=1}^{P} \tag{36}$$

we obtain, as suggested above:

$$\frac{\partial^2 y}{\partial \mathbf{x^2}} \cdot \mathbf{s} = \left[ \left( \frac{\partial^2 v_Q^{(\text{fin})}}{\partial \mathbf{v}^{(\mathbf{ini})\mathbf{2}}} \cdot \mathbf{w}^{(\mathbf{ini})} \right)_p \right]_{p=1}^{P} = \\ = \left[ \mathfrak{d}\overline{v}_p^{(\text{ini})} \right]_{p=1}^{P} . \tag{37}$$

Since each column of an arbitrary matrix can be expressed in the form of a matrix-vector product an appropriately *vectorized* variant of the Hessian algorithm theoretically yields the entire Hessian matrix $\partial^2 y / \partial \mathbf{x^2}$. However, given that $f$ is highly multivariate (i. e. $P \gtrsim 1000$), the practicability of computing $\partial^2 y / \partial \mathbf{x^2}$ almost certainly depends on $\partial^2 y / \partial \mathbf{x^2}$ being sufficiently sparse and, subsequently, on the availability of adequate sparsity information. As a matter of fact, the concept of algorithmic differentiation does include methods, known as "automatic sparsity detection", to efficiently determine sparsity patterns of large Jacobian and Hessian matrices. A further discussion of this advanced topic, though, goes beyond the scope of our introductory article, so that we refer the more interested reader to the respective literature (e. g. [24], [10]) at this point.

## III. An Instructive Example

Having briefly discussed the basic mathematics of algorithmic differentiation, we now illustrate the concept of algorithmic differentiation by considering the *Tikhonov regularization* [25], a simple model often used for regularization of noisy signals. The energy functional associated with the first order regularized model is given by

$$E = \int_\Omega (u - g)^2 \, \mathrm{d}\Omega + \alpha \int_\Omega |\boldsymbol{\nabla} u|^2 \, \mathrm{d}\Omega \tag{38}$$

where $g$ is the given signal (image), $u$ is the unknown, regularized image and $\alpha$ is a free parameter which controls the amount of regularization. The first term is a $L_2$ data fidelity term. The second term is a smoothness term that penalizes high variations in $u$. We point out, that the first-order model can be extended to the full Tikhonov model by additionally penalizing higher-order derivatives in the smoothness term. In this case, the Tikhonov model can be used to implement Gaussian convolution [26].

### A. Implementation of Tikhonov Regularization

Before we start to implement the Tikhonov model, we briefly recall the notation used for image discretization which will be used throughout the paper. We are using a 2D Cartesian grid which is defined as $\{\mathbf{p}_{i,j} = (x_i, y_j) \,|\, 1 \leq i \leq m, \, 1 \leq j \leq n\}$. For convenience, we are using a uniform grid, for which all the subintervals $\triangle x = [x_{i+1} - x_i]$ and $\triangle y = [y_{j+1} - y_j]$ are equal in size. By definition, the use of a Cartesian grid implies a rectangular domain $\Omega = [x_1, x_m] \times [y_1, y_n]$.

Functional (38) can easily be discretized by replacing the integral terms by sums, giving

$$E \approx \sum_{i=1}^{m} \sum_{j=1}^{n} \Big[ (u_{i,j} - g_{i,j})^2 + \alpha \, |\boldsymbol{\nabla} u_{i,j}|^2 \Big] \tag{39}$$

For discretization of the gradient operator we use a simple first order forward differencing scheme defined by

$$|\boldsymbol{\nabla} u_{i,j}|^2 \approx (\delta_x u_{i,j})^2 + (\delta_y u_{i,j})^2 \ , \tag{40}$$

where

$$\begin{aligned} \delta_x u_{i,j} &= \frac{u_{i+1,j} - u_{i,j}}{\Delta x} \\ \delta_y u_{i,j} &= \frac{u_{i,j+1} - u_{i,j}}{\Delta y} \ . \end{aligned} \tag{41}$$

Without loss of generality and for simplicity we may assume $\triangle x = \triangle y = 1$.

Algorithm 1 shows the implementation of the discrete functional (39). Since the variational problem solely refers to $u$, we consider $\alpha$ and $g$ as independent constants, but in general all parameters can be treated as variables. For simplicity, boundary conditions have been omitted, but they could easily be added to the implementation. Clearly, the boundary conditions would then also be included in the derivative algorithms. As implied in section II-A, we have divided the algorithm into three blocks of statements.

**Algorithm 1** Tikhonov Regularization (TR)

```
Constants: $\alpha$, $\mathbf{g}$
Variables: $\mathbf{u}$, $\boldsymbol{\delta}$, $E = 0$
  for $(i, j) = (1, 1)$ to $(m, n)$ do
      ----------------------------------------
      $\delta_1 = u_{i+1,j} - u_{i,j}$
      $\delta_2 = u_{i,j+1} - u_{i,j}$
      ----------------------------------------
      $\delta_3 = \delta_1^2 + \delta_2^2$
      ----------------------------------------
      $E = E + (u_{i,j} - g_{i,j})^2 + \alpha\delta_3$
      ----------------------------------------
  end for
Result: $E$
```

### B. Tangent-Linear Tikhonov Regularization

Algorithm 2 shows the tangent-linear variant of Algorithm 1. According to (16), we obtain the tangent-linear algorithm by gradually applying the differential operator $\mathfrak{d}$ (indicated by the prefix *tl_*) to each statement of the basic algorithm. Given that **for**-loops are just short notations for repeating blocks of statements, they are not changed in the tangent-linear code. In fact, the three blocks of Algorithm 1 transform to the corresponding blocks of Algorithm 2. Since $E$ depends on the variables $\delta_1$ and $\delta_2$ in a non-linear fashion, recomputation of these variables is necessary before the second block of the tangent-linear algorithm can actually be computed.

As indicated in (20), the output of Algorithm 2 basically depends on the initialization of $tl\_E$ and $\mathbf{tl\_u}$. Since the goal is to find those $u_{i,j}$ which minimize $E$, we are interested in computing the partial derivatives $\partial E/\partial u_{i,j}$. Thus we obtain $\partial E/\partial u_{i,j}$ from the tangent-linear algorithm by initializing $tl\_E = 0$ and $\mathbf{tl\_u}$ with the unit vector co-directional with $u_{i,j}$. This implies, that the tangent-linear algorithm has to be executed $mn$ times to compute the entire gradient with respect to $u$. Due to this high computational costs, we do not use the tangent-linear algorithm to compute the gradient. However, the tangent-linear algorithm is the most fundamental one and serves as a basis for further derivative algorithms.

```
Algorithm 2 Tangent-Linear TR
Constants: $\alpha,\ \mathbf{g}$
Variables: $\mathbf{u},\ \boldsymbol{\delta},\ \mathbf{tl\_u} = \hat{\mathbf{d}},\ \mathbf{tl\_}\boldsymbol{\delta},\ tl\_E = 0$
  for $(i,j) = (1,1)$ to $(m,n)$ do
      $tl\_\delta_1 = tl\_u_{i+1,j} - tl\_u_{i,j}$
      $tl\_\delta_2 = tl\_u_{i,j+1} - tl\_u_{i,j}$
      $\delta_1 = u_{i+1,j} - u_{i,j}$
      $\delta_2 = u_{i,j+1} - u_{i,j}$

      $tl\_\delta_3 = 2\delta_1 \cdot tl\_\delta_1 + 2\delta_2 \cdot tl\_\delta_2$

      $tl\_E = tl\_E + 2\left(u_{i,j} - g_{i,j}\right) \cdot tl\_u_{i,j} + \alpha \cdot tl\_\delta_3$
  end for
Result: $tl\_E\ (\partial E/\partial d)$
```

### C. Adjoint Tikhonov Regularization

Algorithm 3 shows the adjoint variant of Algorithm 1. It is obtained by applying the rules developed in section II-C. As the transposition of the global Jacobian involves an inversion of the information flow the adjoint algorithm is generated by gradually evaluating each adjoint block of the reversed basic algorithm. In other words, deriving the adjoint algorithm starts with processing the last block executed in the basic algorithm. Therefore, let us consider the third block of Algorithm 1. According to (29) we have

$$\begin{bmatrix} u_{i,j} \\ \delta_3 \\ E \end{bmatrix} = \begin{bmatrix} u_{i,j} \\ \delta_3 \\ E + \left(u_{i,j} - g_{i,j}\right)^2 + \alpha\delta_3 \end{bmatrix} . \tag{42}$$

Subsequently, we obtain from (30):

$$\begin{bmatrix} ad\_u_{i,j} \\ ad\_\delta_3 \\ ad\_E \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2\left(u_{i,j} - g_{i,j}\right) \\ 0 & 1 & \alpha \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} ad\_u_{i,j} \\ ad\_\delta_3 \\ ad\_E \end{bmatrix} . \tag{43}$$

The adjoint statements are obtained by simply writing down the equation system of (43). In fact, the first three lines of the adjoint algorithm correspond to those equations. In section II-C it was indicated that the adjoint algorithm is preferable for computing gradients because all the partial derivatives $\partial E/\partial u_{i,j}$ are computed at once. For this purpose, the adjoint algorithm is initialized by $ad\_E = 1$ and $\mathbf{ad\_u} = \emptyset$ (see (25)).

```
Algorithm 3 Adjoint TR
Constants: $\alpha,\ \mathbf{g}$
Variables: $\mathbf{u},\ \boldsymbol{\delta},\ \mathbf{ad\_u} = \emptyset,\ \mathbf{ad\_}\boldsymbol{\delta} = \emptyset,\ ad\_E = 1$
  for $(i,j) = (m,n)$ to $(1,1)$ do
      $ad\_u_{i,j} = ad\_u_{i,j} + 2\left(u_{i,j} - g_{i,j}\right) \cdot ad\_E$
      $ad\_\delta_3 = ad\_\delta_3 + \alpha \cdot ad\_E$
      $ad\_E = ad\_E$
      $\delta_1 = u_{i+1,j} - u_{i,j}$
      $\delta_2 = u_{i,j+1} - u_{i,j}$

      $ad\_\delta_1 = ad\_\delta_1 + 2\delta_1 \cdot ad\_\delta_3$
      $ad\_\delta_2 = ad\_\delta_2 + 2\delta_2 \cdot ad\_\delta_3$
      $ad\_\delta_3 = 0$

      $ad\_u_{i,j} = ad\_u_{i,j} - ad\_\delta_1 - ad\_\delta_2$
      $ad\_u_{i+1,j} = ad\_u_{i+1,j} + ad\_\delta_1$
      $ad\_u_{i,j+1} = ad\_u_{i,j+1} + ad\_\delta_2$
      $ad\_\delta_1 = 0$
      $ad\_\delta_2 = 0$
  end for
Result: $\mathbf{ad\_u}\ (\partial E/\partial \mathbf{u})$
```

### D. Equivalence of the Adjoint Variant and the Euler-Lagrange Equation

By the calculus of variations, the first order variation of (38) is given by

$$2\left(u - g\right) - 2\alpha\Delta u = 0 \ , \tag{44}$$

with Neumann boundary condition

$$\frac{\partial u}{\partial n} = 0 \ . \tag{45}$$

When analyzing the behaviour of Algorithm 3, it can be seen that the information at a certain position $(i,j)$ contributes to $ad\_u_{i,j}$, $ad\_u_{i+1,j}$ and $ad\_u_{i,j+1}$. Fig. 1 illustrates the information flow of the adjoint algorithm. Summing up the individual contributions, the adjoint variable $ad\_u$ at position $(i,j)$ yields

$$\begin{aligned} ad\_u_{i,j} = {} & 2\left(u_{i,j} - g_{i,j}\right) + \\ & \alpha\left(-\delta_x u_{i,j} - \delta_y u_{i,j} + \delta_x u_{i-1,j} + \delta_y u_{i,j-1}\right) \end{aligned} \tag{46}$$

Using the equivalence of the one-sided finite differences for adjacent grid points,

$$\begin{aligned} \delta_x u_{i-1,j} &= u_{i,j} - u_{i-1,j} \\ \delta_y u_{i,j-1} &= u_{i,j} - u_{i,j-1} \end{aligned} \tag{47}$$

(46) simplifies to

$$ad\_u_{i,j} = 2\left(u_{i,j} - g_{i,j}\right) - 2\alpha\Delta u_{i,j} \ , \tag{48}$$

where $\Delta u_{i,j} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}$ is a second order approximation of the Laplace operator $\Delta u_{i,j}$.

Comparing (48) with (44), one can see that the output of the adjoint model is equivalent to a second order discretization of the corresponding Euler-Lagrange equation. We point out that this equivalence merely holds for linear problems such as (38). For nonlinear problems, the numerical schemes would be completely different.

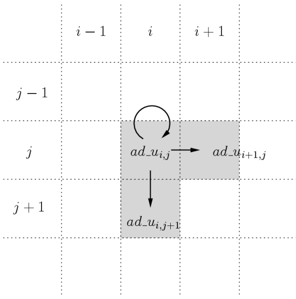


Fig. 1. Information flow at a single pixel in the adjoint model for Tikhonov Regularization.

### *E. Hessian Tikhonov Regularization*

Algorithm 4 shows the Hessian variant of the basic algorithm. It is obtained by deriving the tangent-linear algorithm of the adjoint algorithm. The initialization concept of the Hessian algorithm is equivalent to that of the tangent-linear algorithm. The second-order derivatives $\partial^2 E/\partial u_{i,j} \partial u_{k,l}$ (corresponding to one column of the Hessian matrix) are therefore obtained by initializing $\mathbf{tl\_u}$ with the unit vector co-directional with $u_{k,l}$, for a specific $k \in [1, m]$ and $l \in [1, n]$. Thus, the complete Hessian is obtained by running the Hessian algorithm $mn$ times. Clearly, for large problems, the computation of the complete Hessian matrix can be quite expensive in time and memory. However, for Newton-type line-search purposes [23] or Hessian-free Newton methods [22]), it is only necessary to compute the product of the Hessian matrix with some direction vector $\mathbf{s}$. For this purpose the Hessian algorithm is initialized by $\mathbf{tl\_u} = \mathbf{s}$ and the result is obtained from a single call of the Hessian algorithm (see (35)).

**Algorithm 4** Hessian TR

```
Constants: $\alpha,\ \mathbf{g}$
Variables: $\mathbf{u},\ \boldsymbol{\delta},\ \mathbf{ad\_\delta} = \emptyset,\ ad\_E = 1,\ \mathbf{tl\_u} = \mathbf{d},$
           $\mathbf{tl\_\delta},\ \mathbf{he\_u} = \emptyset,\ \mathbf{he\_\delta} = \emptyset,\ he\_E = 0$
  for $(i,j) = (m,n)$ to $(1,1)$ do
      $he\_u_{i,j} = he\_u_{i,j} + 2\,(u_{i,j} - g_{i,j}) \cdot he\_E +$
      $+ 2ad\_E \cdot tl\_u_{i,j}$
      $he\_\delta_3 = he\_\delta_3 + \alpha \cdot he\_E$
      $he\_E = he\_E$
      $tl\_\delta_1 = tl\_u_{i+1,j} - tl\_u_{i,j}$
      $tl\_\delta_2 = tl\_u_{i,j+1} - tl\_u_{i,j}$
      $ad\_\delta_3 = ad\_\delta_3 + \alpha \cdot ad\_E$
      $\delta_1 = u_{i+1,j} - u_{i,j}$
      $\delta_2 = u_{i,j+1} - u_{i,j}$

      $he\_\delta_1 = he\_\delta_1 + 2\delta_1 \cdot he\_\delta_3 + 2ad\_\delta_3 \cdot tl\_\delta_1$
      $he\_\delta_2 = he\_\delta_2 + 2\delta_2 \cdot he\_\delta_3 + 2ad\_\delta_3 \cdot tl\_\delta_2$
      $he\_\delta_3 = 0$

      $he\_u_{i,j} = he\_u_{i,j} - he\_\delta_1 - he\_\delta_2$
      $he\_u_{i+1,j} = he\_u_{i+1,j} + he\_\delta_1$
      $he\_u_{i,j+1} = he\_u_{i,j+1} + he\_\delta_2$
      $he\_\delta_1 = 0$
      $he\_\delta_2 = 0$
  end for
Result: $\mathbf{he\_u}\ \left((\partial^2 E/\partial \mathbf{u^2}) \cdot \mathbf{d}\right)$
```

## IV. Algorithmic Differentiation for Vision Problems

In order to demonstrate the wide applicability of algorithmic differentiation, we apply the proposed methods to three commonly encountered vision problems: Denoising (image decomposition), segmentation and stereo. For minimization of the functionals, we use a common optimization framework consisting of standard algorithms such as the method of steepest descent, the conjugate gradient method and the quasi-Newton method (see [23]).

### *A. Denoising*

Denoising is commonly used for image restoration which is a significant task in early vision. Total variation minimizing models [27] have become very popular in edge preserving image denoising. Over the years the original formulation of Rudin, Osher and Fatemi [28] has been modified in various ways to improve its performance and has been extended to many other tasks such as image decomposition [29], inpainting [30], and blind deconvolution [31].

The total variation denoising (TVD) model can be expressed as the minimizer of the following energy functional

$$E_{\text{TVD}} = \frac{1}{2\alpha} \int_\Omega (u-g)^2 \, \mathrm{d}\Omega + \int_\Omega |\boldsymbol{\nabla} u| \, \mathrm{d}\Omega \ , \qquad (49)$$

where $g$ is the observed image data, $u$ is the regularized image and $\alpha$ is a parameter which controls the amount of regularization. The first integral term is a data fidelity term, which ensures that the solution $u$ is close to $g$. The second term is the total variation (TV) and penalizes for high variations in $u$. The main property of this term is, that it disfavours oscillations such as noise, but allows for sharp discontinuities (edges).

*1) Discretization:* Comparing (49) to (38), one can see, that the functionals are almost identical except for the norm used in the regularization term. Therefore we only have to reformulate the total variation term which yields

$$|\boldsymbol{\nabla} u_{i,j}| \approx \sqrt{(\delta_x u_{i,j})^2 + (\delta_y u_{i,j})^2} \ , \qquad (50)$$

with $\delta_x u_{i,j}$ and $\delta_y u_{i,j}$ as in (40). Due to the non-differentiability of the total variation term in zero, one has to give some attention to the implementation of this term. The first commonly used method consists of adding a small constant $\epsilon > 0$, resulting in

$$|\boldsymbol{\nabla} u_{i,j}|^{\epsilon} = \sqrt{\left(\delta_x u_{i,j}\right)^2 + \left(\delta_y u_{i,j}\right)^2 + \epsilon^2}\ . \tag{51}$$

The second possibility, which is more in the spirit of algorithmic differentiation is to implement the total variation term using conditional branching.

$$|\boldsymbol{\nabla} u_{i,j}|^{c} = \begin{cases} 0 & \text{if } |\boldsymbol{\nabla} u_{i,j}| = 0 \\ |\boldsymbol{\nabla} u_{i,j}| & \text{else} \end{cases} \tag{52}$$

Using this formulation, the non-differentability is automatically removed when deriving the adjoint code. This also conforms to the underlying optimization problem for which vanishing image gradients do not contribute to the energy functional. The complete discrete energy can consequently be written as

$$E_{\mathrm{TVD}} = \sum_{i=1}^{m}\sum_{j=1}^{n}\left[\frac{1}{2\alpha}\left(u_{i,j} - g_{i,j}\right)^2 + |\boldsymbol{\nabla} u_{i,j}|^{c}\right] , \tag{53}$$

which can be implemented similar to Algorithm 1. In fact, Algorithm 5 (Appendix) shows the basic algorithm implementing (53). Algorithm 6 is the adjoint variant implementing the gradient of $E_{\mathrm{TVD}}$. In Algorithm 7 and Algorithm 8, we also show some excerpts of the C++ implementation and the adjoint code as generated by the preprocessor TAC++ [19]. In order to compute the full gradient, one has to do the following initialization (see also section III-C): `(*_rof_ret_ad) = 1` and `(*u_ad) = [0,...,0]`. Comparing Algorithm 6 with Algorithm 8, one can see that the manually derived code and the automatically generated code are quite similar.

*2) Dual Formulation of the TVD model:* In [32], Chambolle proposed a simple fixpoint algorithm solely based on the dual formulation of the TVD model.

$$\max_{|\mathbf{w}|\leq 1} \int_{\Omega} g\boldsymbol{\nabla}\cdot\mathbf{w}\,\mathrm{d}\Omega - \frac{\alpha}{2}\int_{\Omega}\left(\boldsymbol{\nabla}\cdot\mathbf{w}\right)^2\,\mathrm{d}\Omega\ , \qquad \mathbf{w} = \left(w^x, w^y\right)^T\ , \tag{54}$$

where the solution $\mathbf{w}^*$ of (54) gives the solution $u^* = g - \alpha\boldsymbol{\nabla}\cdot\mathbf{w}^*$ of (53). The advantage of the dual formulation is that it is continuously differentiable in $\mathbf{w}$, but it is important to note that the constraint $|\mathbf{w}| \leq 1$ introduces some additional complexity. However, we demonstrate how algorithmic differentiation can be used to compute the solution of the TVD model using its dual formulation. First, we have to formulate a discrete energy functional which is in fact the negative of (54).

$$E_{\mathrm{TVD}}^{\mathrm{dual}} = \sum_{i=1}^{m}\sum_{j=1}^{n}\left[\frac{\alpha}{2}\left(\mathrm{div}\,\mathbf{w}_{i,j}\right)^2 - g_{i,j}\mathrm{div}\,\mathbf{w}_{i,j}\right] , \tag{55}$$

where $\mathrm{div}\,\mathbf{w}_{i,j} = \left(w^x_{i,j} - w^x_{i-1,j}\right) + \left(w^y_{i,j} - w^y_{i,j-1}\right)$ is the discrete divergence operator. The implementation of (55) is straightforward and its adjoint algorithm can easily be derived. In order to satisfy the constraint $|\mathbf{w}_{i,j}| \leq 1$ we have to project the gradient of (55) (obtained from the adjoint code) to the feasible region (unit circle). For this, we use a limited memory quasi-Newton method for bound constraint optimization [33]. Instead of $|\mathbf{w}_{i,j}| \leq 1$ we require $|w^x_{i,j}| \leq 1$ and $|w^y_{i,j}| \leq 1$ which results in a slightly anisotropic version of the total variation term. The desired solution $u^*_{i,j}$ can be recovered via $u^*_{i,j} = g_{i,j} - \alpha\left(\mathrm{div}\,\mathbf{w}^*\right)_{i,j}$. See [34] for more information about the anisotropic version.

*3) Variants of the TVD model:* In [35], the authors describe the relations between anisotropic diffusion (such as the TVD model) and robust statistics. They showed, that anisotropic diffusion can be seen as a robust estimation procedure, where edges are treated as outliers. A commonly used error norm from the robust statistics literature is Huber's minmax norm [36]:

$$f^{\mathrm{Huber}}(x) = \begin{cases} |x| & \text{if } |x| > \sigma\ , \quad \sigma > 0 \\ \frac{\sigma}{2} + \frac{x^2}{2\sigma} & \text{else.} \end{cases} \tag{56}$$

For large values, this norm is equivalent to the $L_1$ norm (as in the TVD model), but for small values it is designed to be the quadratic $L_2$ norm. The integration of (56) into (53) is very straightforward and can also be implemented using conditional branching.

$$|\boldsymbol{\nabla} u_{i,j}|^{\mathrm{Huber}} = \begin{cases} |\boldsymbol{\nabla} u_{i,j}| & \text{if } |\boldsymbol{\nabla} u_{i,j}| > \sigma \\ \frac{\sigma}{2} + \frac{1}{2\sigma}|\boldsymbol{\nabla} u_{i,j}|^2 & \text{else.} \end{cases} \tag{57}$$

Another interesting robust error norm is due to Geman et al. [37]:

$$f^{\mathrm{Geman}}(x) = \frac{x^2}{1 + \beta x^2}\ , \quad \beta > 0 \tag{58}$$

In [38] it was shown that this non-convex type of error norm (edge preserving boundary function) has the surprising property that it connects two essentially different models: The segmentation model of Mumford and Shah [6] and the anisotropic diffusion equations of Perona and Malik[39]. Again, this error norm can be easily integrated into (53) by

$$|\boldsymbol{\nabla} u_{i,j}|^{\mathrm{Geman}} = \frac{\left(\delta_x u_{i,j}\right)^2 + \left(\delta_y u_{i,j}\right)^2}{1 + \beta\left(\left(\delta_x u_{i,j}\right)^2 + \left(\delta_y u_{i,j}\right)^2\right)} \tag{59}$$

We refer the reader to [40] for a further discussion on edge preserving boundary functions.

*4) Results:* Fig. 2 shows the denoising capability of the TVD model using Algorithm 6. The parameter $\alpha$ was set to 70. For minimization we used a limited memory quasi-Newton method [33]. In practice only a few iterations $(20 - 50)$ are needed to achieve pleasing results.

Fig. 3 illustrates the convergence rates and CPU times (Matlab implementation executed on an AMD Opteron $250$ CPU) of three different optimization algorithms for the TVD model, where $\Delta E'$ is the change of the normalized energy. The first one is based on the adjoint variant of the primal TVD model (53), the second one is based on the adjoint variant of its dual formulation (55), and the last one is the fixed point algorithm of Chambolle [32]. It can be seen that the convergence rates of all three algorithms are comparable.

Fig. 4 shows the comparison of three different robust error norms for a test image. Fig. 4(b) shows the denoised image using the original TVD model. One can observe the staircasing effect of the TVD model. Fig. 4(c) shows the result using Huber's minmax norm. The staircasing effect has been removed, but the edges appear slightly blurred. Fig. 4(d) shows the denoising result using the robust error norm of Geman et

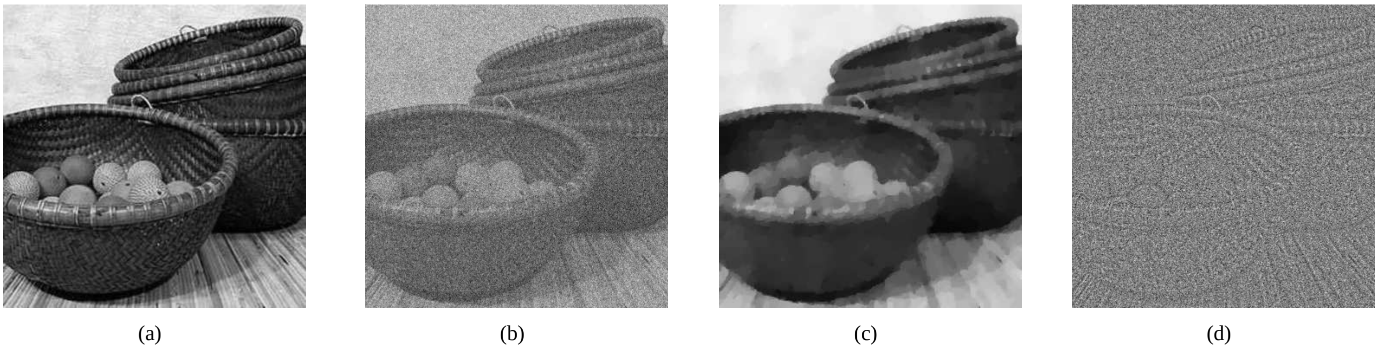


Fig. 2. Denoising capability of the adjoint TVD model. (a) The original image (size = $350 \text{x} 350$, intensity range = $[0, 255]$). (b) The noisy image (Gaussian noise, $\sigma = 50$). (c) The denoised image. (d) The difference image between (b) and (c).

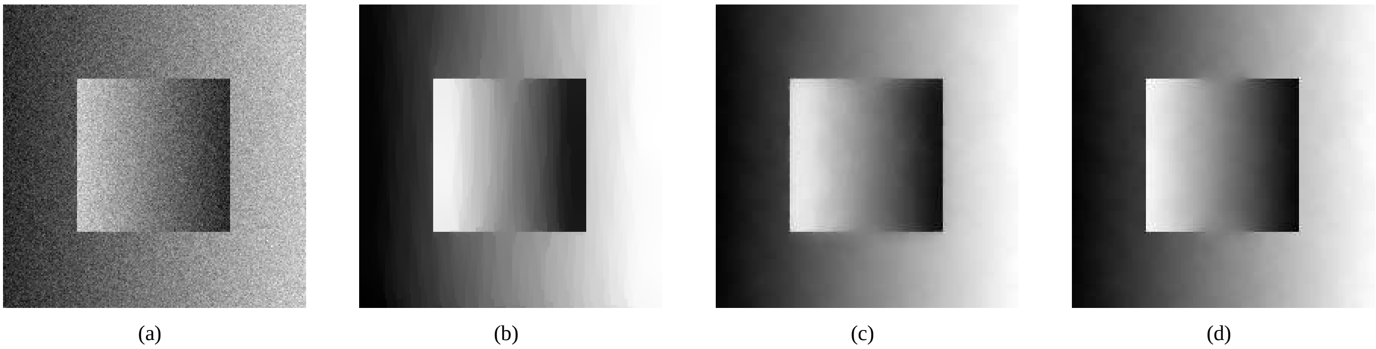


Fig. 4. Comparison of robust error norms. (a) The noisy input image. (b) TVD model ($\alpha = 50$). (c) Huber's minmax norm ($\alpha = 20$, $\sigma = 2$). (d) Edge preserving boundary function of Geman et al ($\alpha = 10$, $\beta = 0.005$).

al. Using this error norm, the image is well denoised and the edges are sharply preserved.

In this example, we considered the total variation denoising model of Rudin, Osher and Fatemi and proposed two minimization algorithms based on its primal and dual formulation. We further showed that different variants can be obtained quite easily by changing a few lines of code. This emphasizes the flexibility of algorithmic differentiation.

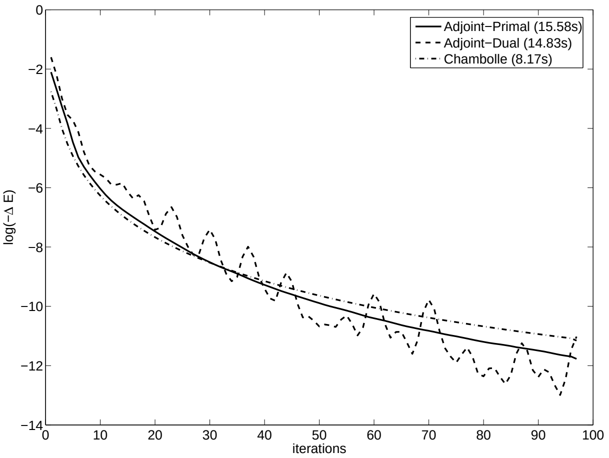


Fig. 3. Convergence rates of different optimization algorithms for the TVD model.

## B. Segmentation

One of the best studied, but still unsolved low level vision problems, the segmentation problem, is a typical example for an inverse problem. The celebrated Mumford-Shah segmentation model [6], [41], which is closely related to statistical estimation via maximum likelihood or maximum a posteriori probabilities [42], [43] has successfully been utilized for several applications e.g. active contour models [44], [45], [46], [47], edge detection [48], [49], image regularization [50], image decomposition [51], inpainting [52], registration [53], and classification [54]. The original Mumford-Shah (MS) segmentation model is defined as

$$E_{\mathrm{MS}} = \int_{\Omega} (u-g)^2 \,\mathrm{d}\Omega \,\alpha \int_{\Omega\setminus\Gamma} |\boldsymbol{\nabla} u|^2 \mathrm{d}\Omega \,+\, \beta\,\mathrm{length}(\Gamma)\ , \tag{60}$$

where $g$ is the observed image, $u$ is a piecewise smooth approximation, $\Gamma$ is a set containing edges in u, $\alpha$ and $\beta$ are tuning parameters. In its original setting, the MS functional is hard to minimize, due to the lack of convexity and regularity of the edge-length term. A solution was given by Ambrosio and Tortorelli [55] via $\Gamma$-convergence, where the edge set (in 2D) is presented by means of a 2D phase field.

We address the *piecewise constant* Mumford-Shah segmentation model (PCMS), which is obtained from the original formulation by $\alpha \to \infty$. In [47] Shen proposed a $\Gamma$-convergence formulation for the piecewise constant case including two regions $\Omega_+$ and $\Omega_-$. The associated energy functional is defined by

$$E_{\mathrm{PCMS}}=\int_\Omega \left(\frac{1+z}{2}\right)^2 (g-c^+)^2 \mathrm{d}\Omega + \int_\Omega \left(\frac{1-z}{2}\right)^2 (g-c^-)^2 \mathrm{d}\Omega + \alpha \int_\Omega \left(9\epsilon|\boldsymbol{\nabla} z|^2 + \frac{(1-z^2)^2}{64\epsilon}\right) \mathrm{d}\Omega \ , \tag{61}$$

where $z$ is the phase field, $\alpha$ is a tuning parameter penalizing the length of the edges, $\epsilon$ is the $\Gamma$-convergence parameter and $c^{\pm}$ are the means of the corresponding regions. For a given partitioning of $\Omega$, the optimal means (see [47], Theorem 1) are given by

$$c^{\pm} = \frac{\int_\Omega (1 \pm z)^2 g \, \mathrm{d}\Omega}{\int_\Omega (1 \pm z)^2 \, \mathrm{d}\Omega} \ . \tag{62}$$

*1) Discretization:* A discretization of (61) and (62) can be realized easily by replacing the integral terms by sums, yielding

$$E_{\mathrm{PCMS}}=\sum_{i=1}^{m}\sum_{j=1}^{n}\Bigg[\left(\frac{1+z_{i,j}}{2}\right)^2 \left(g_{i,j}-c^+\right)^2 + \left(\frac{1-z_{i,j}}{2}\right)^2 \left(g_{i,j}-c^-\right)^2 + \alpha\left(9\epsilon|\boldsymbol{\nabla} z_{i,j}|^2 + \frac{(1-z_{i,j}^2)^2}{64\epsilon}\right)\Bigg] \ , \tag{63}$$

and

$$c^{\pm} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(1 \pm z_{i,j})^2 g_{i,j}}{\sum_{i=1}^{m}\sum_{j=1}^{n}(1 \pm z_{i,j})^2} \ . \tag{64}$$

For simplicity, we use forward differences to approximate the gradient operator (see also section III-A).

$$|\boldsymbol{\nabla} z_{i,j}|^2 \approx \left(\delta_x z_{i,j}\right)^2 + \left(\delta_y z_{i,j}\right)^2 \ . \tag{65}$$

The implementation of (63) is again not challenging.

*2) Results:* Fig. 5 shows the segmentation of a test image using the PCMS model. The phase field was initialized randomly, the parameter $\alpha$ was set to the value of $10^4$ and the parameter $\epsilon$ was set to the value of $10^{-1}$. Fig. 6 illustrates the convergence rates of different optimization algorithms. The first one is a Jacobi method based on linearization of the Euler-Lagrange equation as proposed by Shen [47]. The second one is a limited memory quasi-Newton method (l-BFGS) [33] and the third one is a Hessian-free Newton method (Newton-CG) [22]. For more information about these algorithms we refer the reader to [23]. All algorithms were implemented in Matlab and executed on an AMD Opteron 250 CPU. To run l-BFGS only the adjoint code (first-order derivatives) has to be generated, whereas for Netwon-CG also the Hessian code (second-order derivatives) is needed. As predicted by the theory, l-BFGS and Newton-CG converge quickly, in particular in the proximity of the solution. The Jacobi method steadily converges, but a big number of iterations are necessary to reach the steady state. Table I shows a performance comparison of the different algorithms. Note that after $500$ iterations the Jacobi algorithm has still not converged to the minimizer computed by the other methods.

In this example, we showed that the derivatives obtained from algorithmic differentiation can be applied very straightforward to different minimization algorithms.

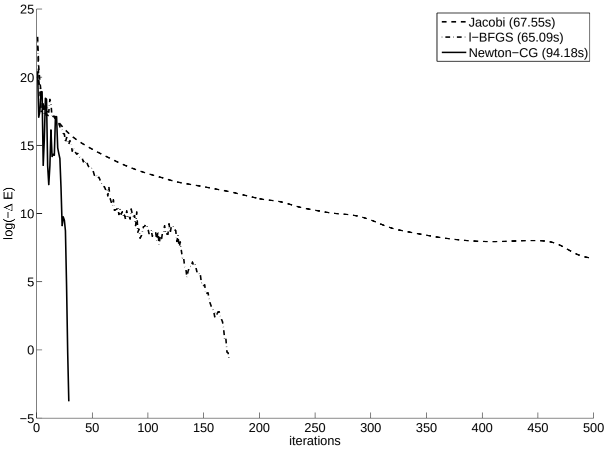


Fig. 6. Convergence rates of different optimization algorithms for the PCMS model and the image shown in Fig. 5.

TABLE I
PERFORMANCE OF DIFFERENT ALGORITHMS FOR THE PCMS MODEL AND THE IMAGE SHOWN IN FIG. 5.

| | Iterations | CPU Time | $E_{\mathrm{PCMS}}$ |
|---|---|---|---|
| Jacobi | 500 | 67.55s | $2.345109 \cdot 10^8$ |
| l-BFGS | 174 | 65.09s | $2.342325 \cdot 10^8$ |
| Newton-CG | 30 | 94.18s | $2.342325 \cdot 10^8$ |

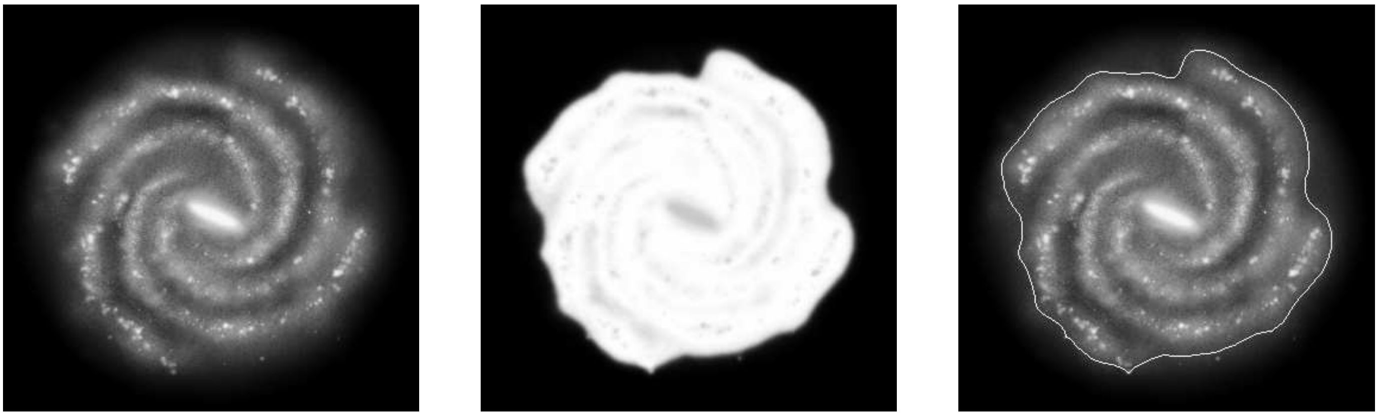
## *C. Stereo*

The recovery of motion from images is a major task of biological and artificial vision systems. In their seminal work, Horn and Schunck [5] studied the so-called *optical flow*, which relates the image intensity at a point and given time to the motion of an intensity pattern. From an application point of view, optical flow plays a major role in 3D-reconstruction [56] and image registration [57] tasks.

The classical optical flow (OF) model of Horn and Schunck which can also be utilized for stereo reconstruction is given by

$$E_{\mathrm{OF}}=\int_\Omega \left(I^1(\mathbf{p}+\mathbf{u}) - I^0(\mathbf{p})\right)^2 \mathrm{d}\Omega + \alpha \int_\Omega |\boldsymbol{\nabla} u_1|^2 + |\boldsymbol{\nabla} u_2|^2 \, \mathrm{d}\Omega \ . \tag{66}$$

$I^0$ and $I^1$ is a stereo image pair, $\mathbf{u} = (u_1(\mathbf{p}), u_2(\mathbf{p}))^T$ is the disparity field and $\alpha$ is a free parameter. The first term is also known as the optical flow constraint. It assumes, that the intensity values of $I^0(\mathbf{p})$ do not change during its motion to $I^1(\mathbf{p}+\mathbf{u})$. The second term penalizes for high variations in $\mathbf{u}$ to obtain smooth disparity fields. Since the OF model penalizes deviations in both terms in a quadratic way, its robustness against outliers such as noise, illumination changes and occlusions is clearly limited. To overcome this, several energy functionals including robust error norms and higher order data terms, have been proposed [58],[56].



(a) (b) (c)

Fig. 5. Two phase segmentation using the PCMS model. (a) Input image (size = 512x512, intensity range = $[0, 255]$). (b) Phase field $z \in [-1, 1]$ after convergence. (c) Zero crossings of the phase field superimposed to the observed image.

Here, we consider the following energy functional for robust optical flow (ROF) computation.

$$E_{\text{ROF}} = \int_\Omega \Psi_D \Big( \big(I^1(\mathbf{p}+\mathbf{u}) - I^0(\mathbf{p})\big)^2 + \beta \big\| \boldsymbol{\nabla} I^1(\mathbf{p}+\mathbf{u}) - \boldsymbol{\nabla} I^0(\mathbf{p}) \big\|^2 \Big) \, \mathrm{d}\Omega + \alpha \int_\Omega \Psi_S \left( \boldsymbol{\nabla} u^T \cdot \mathcal{D} \cdot \boldsymbol{\nabla} u \right) \mathrm{d}\Omega \, . \tag{67}$$

The major differences to the basic OF model are as follows: The functions $\Psi_D$ and $\Psi_S$ are chosen in terms of robust error norms (see also section IV-A.3). We choose the $L_1$ norm $\Psi_D(s) = \Psi_S(s) = \sqrt{s}$ which corresponds total variation regularization. In order to deal with illumination changes we supplement the standard gray value constancy assumption by the constancy of the spatial image gradient [56]. The parameter $\beta$ serves as a weighting factor. Furthermore, we use the image-driven anisotropic regularization technique to avoid smoothing of the disparity field across edges [59]. This method is based on a projection matrix $\mathcal{D}$ perpendicular to $\boldsymbol{\nabla} I^0 = (I^{0,x}, I^{0,y})^T$.

$$\mathcal{D} = \frac{1}{|\boldsymbol{\nabla} I^0|^2 + 2\gamma^2} \begin{bmatrix} (I^{0,y})^2 + \gamma^2 & -I^{0,x} I^{0,y} \\ -I^{0,x} I^{0,y} & (I^{0,x})^2 + \gamma^2 \end{bmatrix} = \begin{bmatrix} \mathcal{D}^a & \mathcal{D}^b \\ \mathcal{D}^b & \mathcal{D}^c \end{bmatrix} , \tag{68}$$

where $\gamma$ is a small parameter which prevents the matrix from getting singular and additionally controls the diffusivity of the matrix.

*1) Discretization:* For notational simplicity, we only deal with the normal case, i.e. with displacements in horizontal direction and thus, $\mathbf{u} = (u(\mathbf{p}), 0))^T$. The extension to the full model is straightforward.

All spatial derivatives are approximated using forward differences $\delta_x$ and $\delta_y$ (see also (41)). Since the displacement field $\mathbf{u}$ is real-valued, we have to do some interpolation to approximate the subpixel values of $I^1$ and $\boldsymbol{\nabla} I^1 = (I^{1,x}, I^{1,y})^T$, respectively. Therefore, we use a quadratic interpolation scheme which is defined e.g. for $I^1$ as

$$P^a_{i,j} = I^1[i + u_{i,j}, j] = a_0 + a_1 \tilde{u} + a_2 \tilde{u}^2 \, , \tag{69}$$

where $a_0$, $a_1$, and $a_2$ are the coefficients of the local quadratic polynomial $P^a_{i,j}$ and $\tilde{u}$ is the displacement of $\mathbf{u}$ relative to the origin of the polynomial. The coefficients of the polynomial are given by

$$a_0 = I^1_{\tilde{i},j} \, , \tag{70}$$

$$a_1 = \frac{I^1_{\tilde{i}+1,j} - I^1_{\tilde{i}-1,j}}{2} \, , \tag{71}$$

$$a_2 = \frac{I^1_{\tilde{i}+1,j} + I^1_{\tilde{i}-1,j}}{2} - I^1_{\tilde{i},j} \, , \tag{72}$$

where $\tilde{i} = i + [u_{i,j}]$ and $\tilde{u} = u_{i,j} - [u_{i,j}]$; the expression $[u_{i,j}]$ denotes a rounding of $u_{i,j}$ to its nearest integer value. In the sequel, the same scheme is applied to construct the local quadratic polynomials $P^b_{i,j}$ with respect to $I^{1,x}$ and $P^c_{i,j}$ with respect to $I^{1,y}$.

Applying the discretization rules to (67) one finally arrives at

$$E_{\text{ROF}} = \sum_{i=1}^{m} \sum_{j=1}^{n} \Big[ e^D_{i,j} + \alpha \, e^S_{i,j} \Big] \, , \tag{73}$$

where

$$e^D_{i,j} = \sqrt{ \left( I^0_{i,j} - P^a_{i,j} \right)^2 + \beta \left( I^{0,x}_{i,j} - P^b_{i,j} \right)^2 + \beta \left( I^{0,y}_{i,j} - P^c_{i,j} \right)^2 } \tag{74}$$

and

$$e^S_{i,j} = \sqrt{ (\delta_x u_{i,j})^2 \mathcal{D}^a_{i,j} + 2 (\delta_x u_{i,j})(\delta_y u_{i,j}) \mathcal{D}^b_{i,j} + (\delta_y u_{i,j})^2 \mathcal{D}^c_{i,j} } \tag{75}$$

*2) Results:* Unlike to the previous examples, the minimization of (73) requires a coarse to fine strategy since the energy functional has many local minima. A usual approach is to generate an image pyramid and to solve the minimization problem on each level. Beginning with the coarsest level, the solution is then propagated as initialization to the next finer

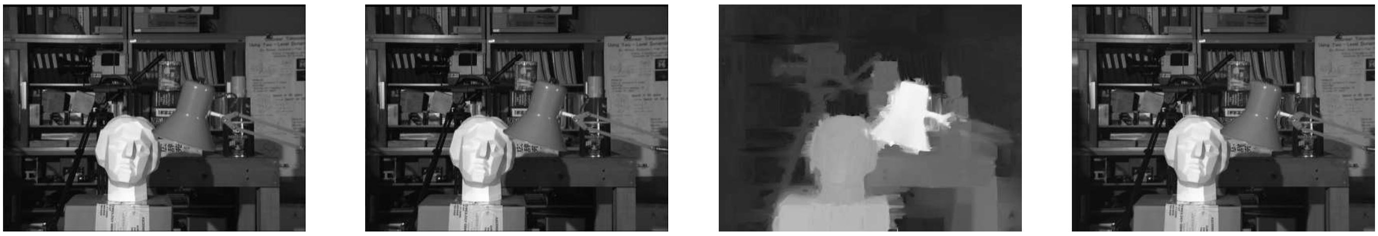


Fig. 7. Recovery of depth information from a stereoscopic image pair using the ROF model. (a) Left image. (b) Right image. (c) Disparity field after optimization. (d) Warped right image.

level until the finest level is reached. More information about such multiscale frameworks can be found in [60].

For optimization, we used a quasi-Newton method [33] which requires only gradient information. Thus we implemented the discretized energy functional (73) and derived the adjoint variant. We built an image pyramid containing 6 levels and ran the optimization algorithm on each level until convergence was reached. On the coarsest level, the disparity field was initialized to zero and the following parameter settings were used: $\alpha = 100.0$, $\beta = 1.0$, and $\gamma = 1.0$. Fig. 7 shows the results of the ROF model applied to a stereoscopic image pair. Fig. 7(c) shows the successful recovery of the depth information.

This example shows that algorithmic differentiation can also be successfully applied when some warping (interpolation) is included in the energy functional. It is also representative for the large class of registration tasks which always include some metric calculated on warped images.

## V. Conclusion

This article introduced the mathematical concept of algorithmic differentiation to the field of computer vision. For a long time, algorithmic differentiation has been used in meteorology, where models are usually highly complex and not representable in a closed form [12]. Focusing on a well-defined class of optimization problems, we briefly discussed the basic mathematics of algorithmic differentiation. We defined three derivative algorithms (i. e. the tangent-linear, adjoint, and Hessian algorithms) and illustrated their potential applicability. In computer vision, many problems can be formulated as minimization of appropriate energy functionals. Actually, we considered three common such problems (namely denoising, segmentation, and stereo) and demonstrated that algorithmic differentiation is well suited to deriving flexible and efficient algorithms for their solution.

Since our article is intended to serve as an introduction, we basically performed algorithmic differentiation by hand. We mentioned, though, several automatic differentiation (AD) tools [11] and presented some adjoint code generated by the preprocessor TAC++. It should be pointed out that the proposed approach is just a *means* to obtain numerical solutions of variational problems, it does not, *per se*, produce novel solutions. Nevertheless, more sophisticated numerical schemes showing better convergence rates and accuracy can be developed quite intuitively. Moreover, the implementation of the derivatives is readily available by the implementation of the energy functional which helps in saving time and avoiding errors.

A major capability of algorithmic differentiation is to identify sparsity patterns of large Jacobian and Hessian matrices [10]. Consequently, given that vision problems are typically large and sparse, algorithmic differentiation could be particularly relevant to reducing the costs of storage and computational time.

After all, we hope that our work will contribute to the steadily growing field of variational methods in computer vision.

## Appendix
## Algorithms

**Algorithm 5** Total Variation Denoising (TVD)

```
for $(i,j) = (1,1)$ to $(m,n)$ do
    $\delta_1 = u_{i+1,j} - u_{i,j}$
    $\delta_2 = u_{i,j+1} - u_{i,j}$
    $\delta_3 = \delta_1^2 + \delta_2^2$
    $E_{\text{TVD}} = E_{\text{TVD}} + \frac{1}{2\alpha}\left(u_{i,j} - g_{i,j}\right)^2$
    if $\delta_3 > 0$ then
        $E_{\text{TVD}} = E_{\text{TVD}} + \sqrt{\delta_3}$
    end if
end for
```

**Algorithm 6** Adjoint TVD

```
for $(i,j) = (m,n)$ to $(1,1)$ do
    $\delta_1 = u_{i+1,j} - u_{i,j}$
    $\delta_2 = u_{i,j+1} - u_{i,j}$
    $\delta_3 = \delta_1^2 + \delta_2^2$
    if $\delta_3 > 0$ then
        $ad\_\delta_3 = \frac{1}{2\sqrt{\delta_3}}$
    end if
    $ad\_u_{i,j} = ad\_u_{i,j} + \frac{1}{\alpha}\left(u_{i,j} - g_{i,j}\right)$
    $ad\_\delta_1 = 2\delta_1 \cdot ad\_\delta_3$
    $ad\_\delta_2 = 2\delta_2 \cdot ad\_\delta_3$
    $ad\_u_{i,j} = ad\_u_{i,j} - ad\_\delta_1 - ad\_\delta_2$
    $ad\_u_{i+1,j} = ad\_u_{i+1,j} + ad\_\delta_1$
    $ad\_u_{i,j+1} = ad\_u_{i,j+1} + ad\_\delta_2$
end for
```

**Algorithm 7** TVD - C++ implementation

```
double rof(int m, int n, double *u0,
           double*u, double alpha) {
  ...
  // sum up data fidelity term
  e1 = 0.0;
  for (i = 0; i < size; ++i) {
    v = *it_u0++ - *it_u++;
    e1 += v*v;
  }
  ...
  // sum up total variation term
  e2 = 0.0;
  for (y = 0; y < n-1; ++y) {
    for (x = 0; x < m-1; ++x) {
      u_x = *it_dux++ - *it_u;
      u_y = *it_duy++ - *it_u++;
      grad_norm_u = sqrt(u_x*u_x + u_y*u_y);
      if (grad_norm_u > 0) {
        e2 += grad_norm_u;
      }
    }
  ...
  // return energy
  return (e1/(2.0*alpha) + e2);
}
```

**Algorithm 8** Adjoint TVD - generated by TAC++

```
void rof_ad(..., double *_rof_ret_ad, ...,
           double *u_ad, ...) {
  ...
  e1_ad+=*_rof_ret_ad/(2.0*alpha);
  e2_ad+=*_rof_ret_ad;
  ...
  for( y=n-1-1; y >= 0; --y ) {
    ...
    for( x=0; x < m-1; ++x ) {
      u_x=*it_dux-*it_u; u_y=*it_duy-*it_u;
      grad_norm_u=sqrt(u_x*u_x+u_y*u_y);
      if( grad_norm_u > 0 ) {
        grad_norm_u_ad+=e2_ad;
      }
      u_x_ad+=grad_norm_u_ad*(2*u_x*
            (0.5F/sqrt(u_x*u_x+u_y*u_y)));
      u_y_ad+=grad_norm_u_ad*(2*u_y*
            (0.5F/sqrt(u_x*u_x+u_y*u_y)));
      grad_norm_u_ad=0;
      *it_duy_ad+=u_y_ad;
      *it_u_ad+=u_y_ad*-1;
      u_y_ad=0;
      *it_dux_ad+=u_x_ad;
      *it_u_ad+=u_x_ad*-1;
      u_x_ad=0;
      ...
    }
    ...
  }
  e2_ad=0;
  ...
  for( i=0; i < size; ++i ) {
    v=*it_u0-*it_u;
    v_ad+=e1_ad*(2*v);
    *it_u_ad+=v_ad*-1;
    v_ad=0;
    ...
  }
  e1_ad=0;
}
```

## Acknowledgments

This research was supported by the Austrian Science Fund (FWF) under the grant P17066-N04 and by Wegener Center young scientist funds. The authors would like to thank Michael Voßbeck, Thomas Kaminski, and Ralf Giering from *Fast*Opt for providing automatically differentiated code (TAC++) and the anonymous reviewers for their constructive comments which helped us to improve this paper.

## References

[1] J. Hadamard, “Sur les problémes aux dérivées partielles et leur signification physique,” *Princeton University Bulletin*, vol. 13, 1902.

[2] M. Bertero, T. Poggio, and V. Torre, “Ill-posed problems in early vision,” in *Proc. of the IEEE*, vol. 76, no. 8, 1988, pp. 869–889.

[3] A. Chambolle, “Inverse problems in image processing and image segmentation: some mathematical and numerical aspects,” Springer Lecture Notes, September 2000.

[4] A. Blake and A. Zisserman, *Visual Reconstruction*. The MIT press, 1987.

[5] B. Horn and B. Schunck, “Determining optical flow,” *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[6] D. Mumford and J. Shah, “Optimal approximation by piecewise smooth functions and associated variational problems,” *Comm. Pure Appl. Math.*, vol. 42, pp. 577–685, 1989.

[7] Y. Shapira, *Solving PDEs in C++: Numerical Methods in a Unified Object-Oriented Approach*, O. Ghattas, Ed. Philadelphia: SIAM, 2006.

[8] A. Griewank and G. Corliess, Eds., *In Automatic Differentiation of Algorithms: Theory, Implementation and Application*. Philadelphia: SIAM, 1991, ch. The use of adjoint equations in numerical modeling of the atmospheric circulation, pp. 69–180.

[9] A. Griewank, “On automatic differentiation,” *Mathematical Programming: Recent Developments and Applications*, pp. 83–108, 1989.

[10] ——, *Evaluating Derivatives: Principles and Techniques of Algorithmic Diferentiation*. in Frontiers in Appl. Math. SIAM, 2000.

[11] “A collection of tools and publications of everything related to automatic differentiation,” www.autodiff.org, July 2006.

[12] X. Zou, F. V. M. Pondeca, and Y. Kuo, “Introduction to adjoint techniques and the MM5 adjoint modeling system,” NCAR Technical Note, Tech. Rep., 1997.

[13] F. Bouttier and P. Courtier, “Data assimilation concepts and methods,” Meteorological Training Course Lecture Series, Tech. Rep., 2002.

[14] A. Griewank, “The chain rule revisited in scientific computing,” *SIAM News*, vol. 24, no. 4, pp. 8–24, 1991.

[15] C. Bischof, A. Carle, P. Khademi, and A. Mauer, “ADIFOR 2.0: Automatic differentiation of Fortran 77 programs,” *IEEE Comput. Sci. Eng.*, vol. 3, no. 3, pp. 18–32, 1996.

[16] R. Giering, T. Kaminski, and T. Slawig, “Applying TAF to a Navier-Stokes solver that simulates an euler flow around an airfoil,” *Future Generation Computer Systems*, vol. 21, no. 8, pp. 1345–1355, 2005.

[17] L. Hascoët, R. Greborio, and V. Pascual, *Computing Adjoints by Automatic Differentiation with TAPENADE*. Springer, 2005, to appear.

[18] A. Griewank, D. Juedes, and J. Utke, “Adol-c: A package for the automatic differentiation of algorithms written in C/C++,” *ACM Trans. On Math. Software*, 1996.

[19] M. Voßbeck, R. Giering, and T. Kaminski, “Automatically generated tangent and adjoint C codes,” Presented at Workshop on Automatic Differentiation, Nice 2005.

[20] R. Giering and T. Kaminski, “Recipes for adjoint code construction,” *ACM Trans. on Math. Softw.*, vol. 24, no. 4, pp. 437–474, 1998.

[21] C. Bischof, “Automatic differentiation, tangent linear models and pseudo-adjoints,” in *High-Performance Computing in the Geosciences*, F.-X. Le Dimet, Ed. Boston: Kluwer Academic Publishers, 1995, vol. 462, pp. 59–80.

[22] J. Nocedal, “Large scale unconstrained optimization,” *The State of the Art in Numerical Analysis*, pp. 311–338, 1997.

[23] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Verlag, 1999.

[24] R. Giering and T. Kaminski, “Automatic sparsity detetection implemented as soruce-to-source transformation,” in *ICCS 2006*, V. A. et al., Ed., vol. 3394. Berlin: Springer, 2006, pp. 591–598.

[25] A. Tikhonov, “On the stability of inverse problems,” *Dokl. Akad. Nauk SSSR*, vol. 39, no. 5, pp. 195–198, 1943.

[26] M. Nielsen, L. Florack, and R. Deriche, “Regularization, scale-space and edge detection filters,” *J. Mathematical Imaging and Vision*, vol. 7, pp. 291–307, 1997.

[27] T. Chan, S. Esedoglu, F. Park, and A. Yip, *The Handbook of Mathematical Models in Computer Vision*. Springer, 2005, ch. Recent Developments in Total Variation Image Restoration.

[28] L. Rudin, S. Osher, and E. Fatemi, “Nonlinear total variation based noise removal algorithms,” *Physica D*, vol. 60, pp. 259–268, 1992.

[29] S. Osher, A. Solé, and L. Vese, “Image decomposition using total variation minimization and the $H^{-1}$ norm,” *Multiscale Modeling and Simulation: A SIAM Interdisciplinary Journal*, vol. 1, no. 3, pp. 349–370, 2003.

[30] T. Chan, S. Kang, and J. Shen, “Euler’s Elastica and curvature-based image inpainting,” *SIAM J. Appl. Math.*, vol. 63, no. 2, pp. 564–592, 2002.

[31] T. Chan and C. Wong, “Total variation blind deconvolution,” *IEEE Trans. Image Processing*, vol. 7, pp. 370–375, 1998.

[32] A. Chambolle, “An algorithm for total variation minimization and applications,” *J. Mathematical Imaging and Vision*, vol. 20, pp. 89–97, 2004.

[33] R. Byrd, P. Lu, and J. Nocedal, “A limited memory algorithm for bound constrained optimization,” *SIAM J. Scientific and Statistical Computing*, vol. 16, no. 6, pp. 1190–1208, 1995.

[34] A. Chambolle, “Total variation minimization and a class of binary MRF models,” in *EMMCVPR 2005*, St. Augustine, FL, USA, November 2005, pp. 136–152.

[35] M. Black, G. Sapiro, D. Marimont, and D. Heeger, “Robust anisotropic diffusion,” *IEEE Trans. Image Processing*, vol. 7, no. 3, 1998.

[36] P. Huber, *Robust Statistics*. New York: Wiley, 1981.

[37] D. Geman and C. Yang, “Nonlinear image recovery with half-quadratic regularization,” *IEEE Trans. Image Processing*, vol. 4, no. 7, pp. 932–946, July 1995.

[38] B. Kawohl, “From Mumford-Shah to Perona-Malik in image processing,” *Math. Meth. Appl. Sci.*, 2003.

[39] P. Perona and J. Malik, “Scale-space and edge detection using anisotropic diffusion,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 7, 1990.

[40] S. Teboul, L. Blanc-Féraud, G. Aubert, and M. Barlaud, “Variational approach for edge-preserving regularization using coupled PDE’s,” *IEEE Trans. Image Processing*, vol. 7, no. 3, pp. 387–397, March 1998.

[41] J. M. Morel and S. Solimini, *Variational Models in Image Segmentation*. Boston: Birkhäuser, 1995.

[42] S. Geman and D. Geman, “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, pp. 721–741, 1984.

[43] D. Mumford, *Geometry Driven Diffusion in Computer Vision*. Kluwer Academic, 1994, ch. The Bayesian rationale for energy functionals, pp. 141–153.

[44] T. Chan and L. Vese, “Active contours without edges,” *IEEE Trans. Image Processing*, vol. 10, no. 2, pp. 266–277, February 2001.

[45] L. Vese, *Geometric Level Set Methods*. Springer, 2003, ch. Multiphase Object Detection and Image Segmentation.

[46] M. Hintermüller and W. Ring, “An inexact newton-CG-type active contour approach for the minimization of the Mumford-Shah functional,” *J. Mathematical Imaging and Vision*, vol. 20, pp. 19–42, 2004.

[47] J. Shen, “$\Gamma$-convergence approximation to piecewise constant Mumford-Shah segmentation,” in *Int’l Conf. Advanced Concepts Intell. Vision Systems*, 2005, pp. 499–506.

[48] A. Brook, R. Kimmel, and N. Sochen, “Variational restoration and edge detection for color images,” *J. Mathematical Imaging and Vision*, vol. 18, pp. 247–268, 2003.

[49] R. March and M. Dozio, “A variational method for the recovery of smooth boundaries,” *Image and Vision Computing*, vol. 15, pp. 705–712, 1997.

[50] W. Vanzella, F. Pellegrino, and V. Torre, “Self adaptive regularization,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 26, no. 6, pp. 804–809, June 2004.

[51] J. Shen, “Piecewise $H^{-1} + H^0 + H^1$ images and the Mumford-Shah-Sobolev model for segmented image decomposition,” *Applied Mathematics Research Express*, vol. 4, pp. 143–167, 2005.

[52] S. Esedoglu and J. Shen, “Digital inpainting based on the Mumford-Shah-Euler image model,” *Eur. J. Applied Mathematics*, vol. 13, pp. 353–370, 2002.

[53] M. Droske, “On variational problems and gradient flows in image processing,” Ph.D. dissertation, Universiät Duisburg-Essen, 2005.

[54] C. Samson, L. Blanc-Féraud, G. Aubert, and J. Zerubia, “A variational model for image classification and restoration,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, no. 5, pp. 460–472, 2000.

[55] L. Ambrosio and V. Tortorelli, “Approximation of functionals depending on jumps by elliptic functionals via $\Gamma$-convergence,” *Comm. Pure Appl. Math.*, vol. 43, pp. 999–1036, 1990.

[56] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert, “Highly accurate optic flow computation with theoretically justified warping,” *Int’l J. Computer Vision*, pp. 141–158, 2006.

[57] B. Vemuri, J. Ye, Y. Chen, and C. Leonard, “A level-set based approach to image registration,” in *IEEE Workshop on Mathematical Methods in Biomedical Image Analysis*, 2000, pp. 86–93.

[58] G. Aubert, R. Deriche, and P. Kornprobst, “Computing optical flow via variational techniques,” *SIAM J. Appl. Math.*, vol. 60, no. 1, pp. 156–182, 1999.

[59] H. Nagel and W. Enkelmann, “An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences,” *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, pp. 565–593, 1986.

[60] M. Lefébure and L. Cohen, “Image registration, optical flow and local rigidity,” *J. Mathematical Imaging and Vision*, vol. 14, pp. 131–147, 2001.

**Thomas Pock** received a BSc and MSc degree in electrical engineering and computer science from Graz University of Technology in 2004. He is currently employed as a research assistant at the Institute for Computer Graphics and Vision at Graz University of Technology, where he is pursuing his PhD degree. His research interests include variational methods, level set methods, inverse problems in image processing and medical image analysis.

**Michael Pock** received an MSc degree in geophysics from the University of Graz in 2004. Currently pursuing a PhD degree in physics, he is a member of the Atmospheric Remote Sensing and Climate System research group of G. Kirchengast at the Wegener Center for Climate and Global Change/University of Graz. His research interests include atmospheric radiative transfer modelling, related inverse problems, algorithmic differentiation and computer arithmetic.

**Horst Bischof** received his MSc and PhD degree in computer science from the Vienna University of Technology in 1990 and 1993, respectively. In 1998 he got his Habilitation (venia docendi) for applied computer science. Currently he is Professor at the Institute for Computer Graphics and Vision at the Technical University Graz, Austria. H. Bischof is Senior researcher at the K+ Competence Center ”Advanced Computer Vision” where he is responsible for research projects in the area of classification. H. Bischof is member of the scientific board of the K+ centers VrVis and KNOW. His research interests include object recognition, visual learning, medical computer vision, neural networks, adaptive methods for computer vision where he has published more than 280 scientific papers.

Horst Bischof was co-chairman of international conferences (ICANN, DAGM), and local organizer for ICPR’96. He was program co-chair of ECCV2006. Currently he is Associate Editor for Pattern Recognition, Computer and Informatics and Journal of Universal Computer Science.

Horst Bischof is currently vice-chair of the Austrian association for pattern recognition. He has received an award from the Pattern Recognition journal in 2002, where the paper ”Multiple Eigenspaces” has been selected as the most original manuscript.