SIAM J. Imaging Sciences
Vol. 11, No. 4, pp. 2665–2691

Variational Image Regularization with Euler's Elastica Using a Discrete Gradient Scheme[*]

Torbjørn Ringholm[†], Jasmina Lazić[‡], and Carola-Bibiane Schönlieb[§]

Abstract. This paper concerns an optimization algorithm for unconstrained nonconvex problems where the objective function has sparse connections between the unknowns. The algorithm is based on applying a dissipation preserving numerical integrator, the Itoh–Abe discrete gradient scheme, to the gradient flow of an objective function, guaranteeing energy decrease regardless of step size. We introduce the algorithm, prove a convergence rate estimate for nonconvex problems with Lipschitz continuous gradients, and show an improved convergence rate if the objective function has sparse connections between unknowns. The algorithm is presented in serial and parallel versions. Numerical tests show its use in Euler's elastica regularized imaging problems and its convergence rate and compare the execution time of the method to that of the iPiano algorithm and the gradient descent and heavy-ball algorithms.

Key words. geometric integration, discrete gradients, Euler's elastica, nonconvex optimization, image inpainting, image denoising

AMS subject classifications. 49M25, 49M37, 65K10, 68W10, 90C26, 90C30

DOI. 10.1137/17M1162354

1. Introduction.

A classic idea for minimizing a differentiable $V : \mathbb{R}^n \to \mathbb{R}$, $n \geq 1$, is considering its gradient flow

$$\dot{\mathbf{u}}(t) = -\nabla V(\mathbf{u}(t)) \tag{1.1}$$

and numerically integrating a solution along it. For example, the gradient descent algorithm can be easily derived from the explicit Euler scheme

$$\mathbf{u}^{k+1} - \mathbf{u}^k = -\tau \nabla V(\mathbf{u}^k),$$

[*] Received by the editors December 21, 2017; accepted for publication (in revised form) September 18, 2018; published electronically November 15, 2018.

http://www.siam.org/journals/siims/11-4/M116235.html

Funding: The work of the third author was supported by the Engineering and Physical Sciences Research Council (EPSRC) grant EP/K009745/1, the Leverhulme Trust project "Breaking the non-convexity barrier," EPSRC grant EP/M00483X/1, EPSRC Centre grant EP/N014588/1, Alan Turing Institute grant TU/B/000071, the Isaac Newton Institute, the Cantab Capital Institute for the Mathematics of Information, and NoMADS (Horizon 2020 RISE project grant). The work of the authors was supported by the European Union's Horizon 2020 research and innovation programme under Marie Sklodowska-Curie grant agreement 691070.

[†] Department of Mathematical Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway (torbjorn.ringholm@ntnu.no).

[‡] MathWorks, Matrix House, 10 Cowley Rd. Cambridge CB4 0HH, UK, and Mathematical Institute, Serbian Academy of Sciences and Arts, Kneza Mihaila 36, Beograd 11001, Serbia (jasmina.lazic@mathworks.co.uk).

[§] Department of Applied Mathematics and Theoretical Physics (DAMTP), University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK (cbs@cam.ac.uk).

where τ is a step size and $\mathbf{u}^k$ an approximation to the value of $\mathbf{u}(k\tau)$. Other schemes can be used, such as Runge–Kutta and multistep methods. A discussion on step size conditions under which algebraically stable Runge–Kutta methods are dissipative can be found in [13]. Even though these classes of ODE integrators are readily available, their use does not appear to have gained much traction in the optimization community.

This disregard may be attributed to a division between the goals of numerical integration and numerical optimization; whereas the ODE integration schemes seek to approximate a solution path of (1.1) as accurately as possible, optimization schemes try to find a stationary point of (1.1) as quickly as possible. The former task requires small time steps while the latter task is generally completed more efficiently the larger the time steps are. Thus, using regular ODE schemes to solve (1.1) is, in general, ineffective. However, in a recent article [29], the authors demonstrate that several well-known efficient optimization methods can be deduced from ODE integration schemes applied to (1.1). Examples include Polyak's heavy-ball method [25], which may also be interpreted as a discretization of a gradient flow with an inertial term, and Nesterov's accelerated gradient method [18]. These methods are equivalent to linear two-step methods with certain choices of step length. Also, the proximal point and proximal gradient methods [20] are shown to correspond to an implicit Euler method and an implicit-explicit scheme, respectively. This gives credibility to the idea that ODE solvers with certain properties may indeed be useful as optimization schemes.

In recent years, new ODE solvers with properties well suited to optimization have emerged. In [12], building on developments in the field of geometric integration, the authors apply discrete gradient schemes to the gradient flow of energy functionals arising from problems in variational image analysis. Discrete gradients, introduced in [11] and further studied in [17], have a property which is interesting from an optimization viewpoint: they are *dissipativity preserving*. When applied to a dissipative ODE such as a gradient flow, the numerical solution is dissipative in the sense that

$$V(\mathbf{u}^{k+1}) \leq V(\mathbf{u}^k).$$

The schemes thus convergence monotonously toward a critical point V^* regardless of the step size used in the numerical integration if V is continuously differentiable [12].

While very efficient solvers exist for convex optimization problems (see, e.g., [8, 21, 31]), the picture is different for nonconvex optimization problems. Considerable effort has been spent on developing efficient schemes for classes of problems with special structure, e.g., problems with one convex but nondifferentiable term and one nonconvex but differentiable term [4, 24]. We will add to this effort by introducing a method based on the Itoh–Abe discrete gradient method [15] that is most effective when the objective function is continuously differentiable with sparsely connected unknowns. Indeed, in [12], the authors use discrete gradient schemes with nonconvex problems in mind, so this paper may be viewed as a continuation of their work.

A problem that fits the format of being nonconvex with sparse connections is variational image analysis using a discretized Euler's elastica functional as a regularizer. Introduced in [22] for deoccluding objects in images, Euler's elastica regularization was further analyzed and applied to inpainting problems in [30] by Chan, Kang, and Shen, who derive the Euler–Lagrange equations for the continuous Euler's elastica functional and solve these via finite difference schemes. This approach is not very computationally efficient, and attempts have been made to create more effective schemes, in particular in [32], where an augmented Lagrangian approach was considered. This approach was later refined in [38], [36], [1], and [37]. Also of note are the approaches in [5] and [9], where convex approximations to the objective function are considered.

The method presented in Algorithm 2.1 resembles the coordinate gradient descent method, except that the gradient is approximated by a discrete gradient. It is derivative free and easy to use with only a step size parameter to choose. The scheme guarantees decrease at each iteration, at the expense of being implicit. A recent survey of coordinate descent algorithms and their convergence can be found in [34]. According to this survey, for coordinate descent methods one can expect $V(x^k) - V^* \in \mathcal{O}(1/k)$ for convex problems, with linear convergence if the problem is strongly convex. In [19], an accelerated coordinate descent method is presented, with $V(x^k) - V^* \in \mathcal{O}(1/k^2)$ for convex problems at the expense of computing a full vector operation for each coordinate update. In the following, we will prove a convergence rate of $\min_{1\leq j\leq k}\left\{\|\nabla V(\mathbf{u}^j)\|^2\right\} \in \mathcal{O}(1/k^{1/2})$ for nonconvex, Lipschitz continuously differentiable problems. In [10], an $\mathcal{O}(1/k)$ convergence rate for convex, smooth problems and linear convergence for problems satisfying the Polyak–Łojasiewicz condition [16] are proved for several discrete gradient algorithms, including the one considered here. In the case of the Itoh–Abe discrete gradient, the rates have an $\mathcal{O}(n^{1/2})$ dependence on the problem size n; in Lemma 3.1 we show that the rates can be improved when V has sparsely connected unknowns. A convergence result for Itoh–Abe type discrete gradient schemes applied to nondifferentiable, nonconvex problems is presented in [27], with applications to parameter estimation problems.

The paper is organized as follows. In the following section, we introduce discrete gradient methods for optimization and discuss the convergence rate and acceleration of a specific discrete gradient-type scheme. We also propose a method for adaptive time steps that allows acceleration of the algorithm for differentiable V, using four additional parameters. In section 3, the Euler's elastica regularization problem is introduced, and parallelization of the discrete gradient algorithm is discussed together with the effect of sparsity in the problem. Section 4 contains numerical experiments concerning the quality of denoising and inpainting, experimental convergence rates, the effect of coordinate ordering and problem size, execution time, and dependence on the initial condition. The final section summarizes the results.

2. Discrete gradient methods.

The task at hand is to minimize a $\mathcal{C}^1$ functional $V : \mathbb{R}^n \to \mathbb{R}$, also called an energy, by solving its gradient flow

$$\dot{\mathbf{u}} = -\nabla V(\mathbf{u}), \tag{2.1}$$

where $\mathbf{u}(t) \in \mathbb{R}^n$ is the unknown and $\dot{\mathbf{u}}(t)$ denotes its time derivative. The reason for this is that V dissipates along the flow of (2.1); if $\mathbf{u}(t)$ solves (2.1), then

$$\frac{\mathrm{d}}{\mathrm{d}t}V(\mathbf{u}(t)) = \langle \dot{\mathbf{u}}, \nabla V(\mathbf{u}(t))\rangle = -\|\nabla V(\mathbf{u}(t))\|^2 \leq 0,$$

where $\|\cdot\|$ denotes the Euclidian norm on $\mathbb{R}^n$. Due to the dissipation, $\mathbf{u}(t)$ approaches a critical point of V as $t \to \infty$ as long as V is bounded from below. In general, V is nonlinear such that numerical schemes must be employed to solve (2.1) until a large stopping time T. This gives rise to different optimization algorithms depending on the scheme used. For example, a forward Euler scheme results in the gradient descent method. The forward Euler method has several drawbacks, one being that choosing too large step sizes results in instability. This necessitates step size selection, which may result in impractically small steps considering that we wish to obtain a stationary point of (2.1). It is therefore of interest to investigate the use of numerical schemes that have lenient step size restrictions or none at all. One such class of schemes is called *discrete gradient* methods. Discrete gradients were introduced in [11] to unite several energy preserving and dissipative ODE solvers under a single label. A seminal paper [17] covers their use as ODE solvers and which ODEs they are applicable to.

Definition 2.1. *Given a differentiable function* $V : \mathbb{R}^n \to \mathbb{R}$, *we say that* $\overline{\nabla} V : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ *is a discrete gradient of* V *if it is continuous and for all* $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$,

$$\left\langle \overline{\nabla} V(\mathbf{u}, \mathbf{v}), \mathbf{v} - \mathbf{u} \right\rangle = V(\mathbf{v}) - V(\mathbf{u}),$$
$$\lim_{\mathbf{v} \to \mathbf{u}} \overline{\nabla} V(\mathbf{u}, \mathbf{v}) = \nabla V(\mathbf{u}).$$

Discrete gradients can be used in schemes to solve (2.1) numerically by computing

$$\mathbf{u}^{k+1} = \mathbf{u}^k - \tau_k \overline{\nabla} V(\mathbf{u}^k, \mathbf{u}^{k+1}), \tag{2.2}$$

where $\tau_k > 0$ is the step size at iteration number k. A key property is that the scheme is dissipating; by Definition 2.1 and the scheme (2.2), we have

$$V(\mathbf{u}^{k+1}) - V(\mathbf{u}^k) = \left\langle \overline{\nabla} V(\mathbf{u}^k, \mathbf{u}^{k+1}), \mathbf{u}^{k+1} - \mathbf{u}^k \right\rangle = -\frac{1}{\tau_k} \|\mathbf{u}^{k+1} - \mathbf{u}^k\|^2. \tag{2.3}$$

Note that the dissipation property holds regardless of the step size τ_k.

Definition 2.1 is quite broad and, as a result, there exist several types of discrete gradients. Two popular choices are the midpoint discrete gradient [11] and the average vector field (AVF) discrete gradient [14]. They give second-order accurate schemes for (2.1) and are suited for solving ODEs precisely. In our case, solving (2.1) as exactly as possible is not the main concern; rather, we need a scheme with cheap time steps and fast convergence toward a minimizer. The schemes obtained using the Gonzalez and AVF discrete gradients are fully implicit in the sense that in general, at each time step of (2.2), a single n-dimensional system of nonlinear equations must be solved. For large n, this is slow since the complexity of solving such a system is typically $\mathcal{O}(n^2)$. Instead, we consider the Itoh–Abe discrete gradient [15], defined componentwise as

$$(\overline{\nabla} V(\mathbf{u}, \mathbf{v}))_l = \frac{V\left(\mathbf{u} + \sum_{j=1}^{l} (v_j - u_j)\mathbf{e}_j\right) - V\left(\mathbf{u} + \sum_{j=1}^{l-1} (v_j - u_j)\mathbf{e}_j\right)}{v_l - u_l},$$

where $\mathbf{e_j}$ denotes the jth standard basis vector. This discrete gradient, while still implicit, has two advantages over the Gonzales and AVF discrete gradients. First, its use in the scheme (2.2) requires the solution of n scalar nonlinear equations per time step, meaning its computational complexity scales as $\mathcal{O}(n)$. Second, it is derivative-free and requires only computations of differences between the objective function with variation in one variable, which may be considerably cheaper than evaluating the objective function itself; consider, for example, the optimization problem

$$\min_{\mathbf{u}\in\mathbb{R}^n} \left\{ V(\mathbf{u}) = \sum_{i=1}^{M} V_i(\mathbf{u}) \right\},$$

where at most N of the V_i depend on a given coordinate, say, u_k. Then, computing the difference $V(\mathbf{u}+\mathbf{e}_k u_k) - V(\mathbf{u})$ amounts to computing at most N values of the V_i, a cost comparable to that of calculating one coordinate derivative of V.

2.1. The algorithm.

The algorithm based on using the Itoh–Abe discrete gradient with fixed step size $\tau_k = \tau$ in (2.2) is presented in Algorithm 2.1. As a stopping criterion we set a tolerance *tol* and stop when $(V(\mathbf{u}^k) - V(\mathbf{u}^{k-1}))/V(\mathbf{u}^0) < tol$. This criterion is economical to evaluate, requiring no evaluation of V since the energy increments are known from the dissipation property (2.3). To solve the nonlinear scalar subproblems defining the β_j^k, we use the Brent–Dekker algorithm [6]. This is a derivative-free method based on a combination of bisection and interpolation algorithms which converges superlinearly if the function whose root is to be found is $\mathcal{C}^1$ near the root. It is the method of choice for scalar root finding problems in [26].

Algorithm 2.1 DG

```
Choose $\tau > 0$, $tol > 0$ and $\mathbf{u}^0 \in \mathbb{R}^n$. Set $k = 0$.
repeat
  $\mathbf{v}_0^k = \mathbf{u}^k$
  for $j = 1, \dots, n$ do
    Solve $\beta_j^k = -\tau(V(\mathbf{v}_{j-1}^k + \beta_j^k \mathbf{e}_j) - V(\mathbf{v}_{j-1}^k))/\beta_j^k$
    $\mathbf{v}_j^k = \mathbf{v}_{j-1}^k + \beta_j^k \mathbf{e}_j$
  end for
  $\mathbf{u}^{k+1} = \mathbf{v}_n^k$
  $k = k + 1$
until $\left(V(\mathbf{u}^k) - V(\mathbf{u}^{k-1})\right)/V(\mathbf{u}^0) < tol$
```

The algorithm can be accelerated through adaptive step sizes. Unlike line search methods, each time step is implicit and so changing τ_k requires a recomputation of $\mathbf{u}^{k+1}$, which can be costly and should be avoided. One way of adapting τ_k, which can be used for differentiable V, is to check conditions similar to the Wolfe conditions [33]. We consider, with constants $c_1 \in (0,1)$ and $c_2 \in (c_1, 1)$, the conditions

$$V(\mathbf{u}^{k+1}) - V(\mathbf{u}^k) \leq c_1 \left\langle \nabla V(\mathbf{u}^k), \mathbf{u}^{k+1} - \mathbf{u}^k \right\rangle, \tag{2.4}$$

$$\left\langle \nabla V(\mathbf{u}^{k+1}), \mathbf{u}^{k+1} - \mathbf{u}^k \right\rangle \geq c_2 \left\langle \nabla V(\mathbf{u}^k), \mathbf{u}^{k+1} - \mathbf{u}^k \right\rangle. \tag{2.5}$$

If condition (2.4) holds, regardless of whether (2.5) holds, then τ_k is increased for the next iteration by a factor $\lambda > 1$. If (2.4) does not hold but (2.5) does, one takes $\tau_{k+1} = \rho\tau_k$,

Algorithm 2.2 DG-ADAPT

```
Choose $\tau_0 > 0$, $tol > 0$, $\rho \in (0,1)$, $\lambda > 1$, $c_1 \in (0,1)$, $c_2 \in (c_1, 1)$, and $\mathbf{u}^0 \in \mathbb{R}^n$.
Set $k = 0$.
repeat
  $\mathbf{v}_0^k = \mathbf{u}^k$
  for $j = 1, \dots, n$ do
    Solve $\beta_j^k = -\tau_k (V(\mathbf{v}_{j-1}^k + \beta_j^k \mathbf{e}_j) - V(\mathbf{v}_{j-1}^k))/\beta_j^k$
    $\mathbf{v}_j^k = \mathbf{v}_{j-1}^k + \beta_j^k \mathbf{e}_j$
  end for
  $\mathbf{u}^{k+1} = \mathbf{v}_n^k$
  if $V(\mathbf{u}^{k+1}) - V(\mathbf{u}^k) \leq c_1 \left\langle \nabla V(\mathbf{u}^k), \mathbf{u}^{k+1} - \mathbf{u}^k \right\rangle$ then
    $\tau_{k+1} = \lambda \tau_k$
  else if $\left\langle \nabla V(\mathbf{u}^{k+1}), \mathbf{u}^{k+1} - \mathbf{u}^k \right\rangle \geq c_2 \left\langle \nabla V(\mathbf{u}^k), \mathbf{u}^{k+1} - \mathbf{u}^k \right\rangle$ then
    $\tau_{k+1} = \rho \tau_k$
  end if
  $k = k + 1$
until $\left(V(\mathbf{u}^k) - V(\mathbf{u}^{k-1})\right)/V(\mathbf{u}^0) < tol$
```

where $\rho \in (0,1)$. If neither condition holds, the step size is not changed. In all cases, the new value $\mathbf{u}^{k+1}$ is accepted. With this approach one obtains step sizes that are adjusted based on prior performance while not wasting previous computations, summed up in Algorithm 2.2. Condition (2.5) provides a lower bound on the step size when ∇V is Lipschitz continuous with Lipschitz constant L. First, since ∇V is Lipschitz, the descent lemma [3, Proposition A.24] provides the estimate

$$V(\mathbf{u}) \leq V(\mathbf{v}) + \langle \nabla V(\mathbf{v}), \mathbf{u} - \mathbf{v} \rangle + \frac{L}{2} \|\mathbf{v} - \mathbf{u}\|^2 \tag{2.6}$$

which holds for all $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$. Combining (2.6) with (2.5) and (2.3) we find

$$\begin{aligned} c_2 \left\langle \nabla V(\mathbf{u}^k), \mathbf{u}^k - \mathbf{u}^{k+1} \right\rangle &\geq \left\langle \nabla V(\mathbf{u}^{k+1}), \mathbf{u}^k - \mathbf{u}^{k+1} \right\rangle \\ &\geq V(\mathbf{u}^k) - V(\mathbf{u}^{k+1}) - \frac{L}{2} \|\mathbf{u}^{k+1} - \mathbf{u}^k\|^2 \\ &= \left(1 - \frac{L}{2}\tau_k\right) \left(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})\right). \end{aligned}$$

Rearranging and applying (2.6) once more, we find

$$\frac{L}{2}\tau_k \left(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})\right) \geq (1 - c_2) \left(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})\right) - \frac{c_2 L}{2} \|\mathbf{u}^{k+1} - \mathbf{u}^k\|^2.$$

Using (2.3) on the last term to eliminate $V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})$ we find the lower bound

$$\tau_k \geq \frac{1 - c_2}{1 + c_2} \frac{2}{L}.$$

2.2. Convergence.

In [12], the authors prove that the iterates of Algorithm 2.1 converge toward a critical point, but they do not estimate the convergence rate. Theorem 2.3 concerns the convergence rate of Algorithm 2.2 in the general case of a nonconvex objective V. A sublinear convergence rate of $\mathcal{O}(1/k)$ for convex problems and a linear convergence rate for problems satisfying the Polyak–Łojasiewicz inequality are given in [10]. These theorems are stated below without proof to support the discussion of numerical results presented in section 4, where better rates than the ones proved in Theorem 2.3 are observed when choosing ϵ in the smoothing of the Euler's elastica regularizer large enough. The following assumption is common to these theorems and Lemma 3.1 in the subsequent section. Here, by coordinate Lipschitz continuity of $f : \mathbb{R}^n \to \mathbb{R}^n$, we mean that for $j = 1, \ldots, n$, $|f_j(\mathbf{x}) - f_j(\mathbf{y})| \leq L_j \|\mathbf{x} - \mathbf{y}\|$.

Assumption 2.2. *The function $V : \mathbb{R}^n \to \mathbb{R}$ is $\mathcal{C}^1$, bounded from below, and coercive. Furthermore, ∇V is Lipschitz with Lipschitz constant L and coordinatewise Lipschitz constants $L_j \in [L_{\min}, L_{\max}]$, and all time steps τ_j lie in $[\tau_{\min}, \tau_{\max}] \subset \mathbb{R}^+$.*

The following proof is inspired by that of [2, Lemma 3.3] and bears resemblance to the classical theorem of Zoutendijk for line search methods [23, Theorem 3.2], with the exception that it does not rely on a Wolfe condition for step sizes and explicitly states a convergence rate.

Theorem 2.3. *If Assumption 2.2 holds, the $\mathbf{u}^k$ produced by Algorithm 2.2 satisfy*

$$\min_{1\leq j\leq k} \left\{\|\nabla V(\mathbf{u}^j)\|^2\right\} \leq \nu \frac{V(\mathbf{u}^0) - V^*}{k}, \qquad \nu = 2L_{\max}^2 \left(\tau_{\max} n + \frac{\tau_{\max}}{L_{\max}^2 \tau_{\min}^2}\right),$$

where $V^* > -\infty$ is a local minimum.

Proof. The coordinatewise Itoh–Abe scheme, with $\beta_l^k = u_l^{k+1} - u_l^k$, reads

$$\beta_l^k = -\tau_k \frac{V\left(\mathbf{v}_l^k\right) - V\left(\mathbf{v}_{l-1}^k\right)}{\beta_l^k},$$

with $\mathbf{v}_l^k := \mathbf{u}^k + \sum_{j=1}^{l} \beta_j^k \mathbf{e}_j$ such that $\mathbf{u}^{k+1} = \mathbf{v}_n^k$. By the triangle inequality,

$$\left|\frac{\partial V}{\partial u_l}(\mathbf{u}^{k+1})\right| \leq \left|\frac{\partial V}{\partial u_l}(\mathbf{u}^{k+1}) - \frac{V(\mathbf{v}_l^k) - V(\mathbf{v}_{l-1}^k)}{\beta_l^k}\right| + \frac{1}{\tau_k}\left|\beta_l^k\right|.$$

Since $V \in \mathcal{C}^1$, the mean value theorem holds, meaning

$$\frac{V(\mathbf{v}_l^k) - V(\mathbf{v}_{l-1}^k)}{\beta_l^k} = \frac{\partial V}{\partial u_l}(\mathbf{v}_{l-1}^k + s\beta_l^k \mathbf{e}_l)$$

for some $s \in (0, 1)$. Hence,

$$\left|\frac{\partial V}{\partial u_l}(\mathbf{u}^{k+1})\right| \leq \left|\frac{\partial V}{\partial u_l}(\mathbf{u}^{k+1}) - \frac{\partial V}{\partial u_l}(\mathbf{v}_{l-1}^k + s\beta_l^k \mathbf{e}_l)\right| + \frac{1}{\tau_k}\left|\beta_l^k\right|. \tag{2.7}$$

Exploiting the coordinatewise Lipschitz continuity of the gradient, we have

$$\left|\frac{\partial V}{\partial u_l}(\mathbf{u}^{k+1})\right| \leq L_l\|\mathbf{u}^{k+1} - \mathbf{v}_{l-1}^k - s\beta_l^k \mathbf{e}_l\| + \frac{1}{\tau_k}\left|\beta_l^k\right|.$$

Squaring this and summing over all coordinates, we get

$$\begin{aligned}\|\nabla V(\mathbf{u}^{k+1})\|^2 &\leq \sum_{l=1}^n \left(L_l\|\mathbf{u}^{k+1} - \mathbf{v}_{l-1}^k - s\beta_l^k \mathbf{e}_l\| + \frac{1}{\tau_k}\left|\beta_l^k\right|\right)^2 \\ &\leq 2L_{\max}^2\left(n + \frac{1}{L_{\max}^2\tau_{\min}^2}\right)\|\mathbf{u}^{k+1} - \mathbf{u}^k\|^2 \\ &\leq \nu\left(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})\right).\end{aligned}$$

We then find

$$k \min_{1\leq j\leq k}\left\{\|\nabla V(\mathbf{u}^j)\|^2\right\} \leq \sum_{j=1}^k \|\nabla V(\mathbf{u}^j)\|^2 \leq \nu\left(V(\mathbf{u}^0) - V(\mathbf{u}^k)\right) \leq \nu\left(V(\mathbf{u}^0) - V^*\right),$$

which concludes the proof. ■

Remark. We can choose a fixed $\tau_k = \tau$ that minimizes ν, yielding

$$\tau = \frac{1}{L_{\max}\sqrt{n}}, \qquad \nu = 4L_{\max}\sqrt{n}.$$

Thus, the complexity of the above bound with respect to the problem size n is $\mathcal{O}(n^{1/2})$. Furthermore, we can obtain bounds of the type $\|\nabla V(\mathbf{u}^{k+1})\|^2 \leq \nu\left(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})\right)$ for other discrete gradients, yielding similar convergence rates. Such bounds are shown in [10, Lemma 5.1].

As with descent methods, the convergence rate improves with additional assumptions on V, in particular assuming that V is convex. We state the following theorems, proved in [10] and inspired by those in [2], for later reference. Similarly to the proof of Theorem 2.3, they are based on bounding $\|\nabla V(\mathbf{u}^{k+1})\|^2 \leq \nu\left(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})\right)$ and so the factor ν appears here as well.

Theorem 2.4. *If Assumption 2.2 holds and V is in addition convex, the iterates $\mathbf{u}^k$ produced by Algorithm 2.2 satisfy, with ν as in Theorem 2.3,*

$$V(\mathbf{u}^k) - V^* \leq \frac{\nu R(\mathbf{u}^0)^2}{k + 2\nu/L}.$$

where V^* is a minimum and $R(\mathbf{u}^0)$ is the diameter of $\{\mathbf{u} \in \mathbb{R}^n | V(\mathbf{u}) \leq V(\mathbf{u^0})\}$.

The next theorem concerns the convergence rate of Assumption 2.2 when V is a PŁ-function, i.e., V satisfies the Polyak–Łojasiewicz inequality with parameter σ,

$$\frac{1}{2}\|\nabla V(\mathbf{u})\|^2 \leq \sigma(V(\mathbf{u}) - V^*).$$

Note that under Assumption 2.2, all strongly convex functions are PŁ-functions [16].

Theorem 2.5. *If Assumption 2.2 holds and V is a PŁ-function, the iterates of Algorithm 2.2 satisfy, with ν as in Theorem 2.3,*

$$V(\mathbf{u}^k) - V^* \leq \left(1 - \frac{2\sigma}{\nu}\right)^k (V(\mathbf{u}^0) - V^*).$$

Remark. The above theorems mean that for convex problems, too, the algorithm has a worst-case complexity of $\mathcal{O}(n^{1/2})$ with respect to the problem dimension n, compared to $\mathcal{O}(n^{3/2})$ for the cyclic coordinate descent algorithm [34] and $\mathcal{O}(n)$ for the expected bounds of stochastic coordinate descent [19]. We shall see in Lemma 3.1 that the complexity can be reduced depending on a sparsity property of V.

3. The Euler's elastica problem.

We will use Algorithm 2.1 for variational image analysis with Euler's elastica regularization. In variational image analysis one repairs a damaged input grayscale image $g : \Omega \to [0,1]$, where $\Omega \subset \mathbb{R}^2$ is often rectangular, by finding an output image $u : \Omega \to [0,1]$ that minimizes a functional

$$V_c(u) = d_c(K_c u, g) + \alpha J_c(u). \tag{3.1}$$

Here, K_c is a forward operator relating u to g, d_c a function measuring the distance between $K_c u$ and g, J_c a regularization functional, and $\alpha > 0$ a constant. The subscript c emphasizes that the functions are continuous; they will later be discretized and renamed. When J_c is the Euler's elastica energy below, α is included in a and b.

The forward operator K_c, which may be linear, is inherent to the problem. For example, when considering an inpainting problem where the goal is to interpolate g in a subset D of the image domain Ω in which there is no given data, one would take K_c as a restriction to $\Omega \backslash D$. Since this leaves $K_c u$ undefined in Ω, the fidelity term should only compare with values of g on $\Omega \backslash D$. This has the effect of maintaining fidelity only in areas where the image is known, at the cost of generating an ill-posed problem due to nonunique solutions. In the denoising problem, where random noise is added to an image in unknown pixels, the usual choice is to take K_c as the identity operator since there is no information about which pixels are damaged.

The terms that differentiate approaches to image analysis are the d_c and J_c functions and the implementation of the K_c operator if applicable. One often takes d_c as an L^p metric, while J_c can be chosen in several ways. A popular choice is the total variation (TV) [28] regularization which, for differentiable u, can be stated as

$$J_{TV}(u) = \int_\Omega |\nabla u| \mathrm{d}\mathbf{x}.$$

In practice, one often wishes to work with a differentiable function after discretizing J, thus using a smoothed version of J_{TV}, with $0 < \epsilon \ll 1$, given as

$$J_{TV_\epsilon}(u) = \int_\Omega |\nabla u|_\epsilon \mathrm{d}\mathbf{x} = \int_\Omega \sqrt{\frac{\partial u}{\partial x}^2 + \frac{\partial u}{\partial y}^2 + \epsilon}\, \mathrm{d}\mathbf{x}.$$

This is the TV$_\epsilon$ regularizer. The Euler's elastica regularizer generalizes J_{TV}, adding a curvature dependent term. It is stated for $\mathcal{C}^2(\Omega)$ functions u as [30]

$$J_c(u) = \int_\Omega \left(a + b \left(\nabla \cdot \frac{\nabla u}{|\nabla u|} \right)^2 \right) |\nabla u| \mathrm{d}\mathbf{x},$$

where $a, b > 0$. We will consider the smoothed version

$$J_\epsilon(u) = \int_\Omega C(u)G(u)\mathrm{d}\mathbf{x}, \quad C(u) = a + b\left(\nabla \cdot \frac{\nabla u}{|\nabla u|_\epsilon}\right)^2, \quad G(u) = |\nabla u|_\epsilon. \tag{3.2}$$

where $C(u)$ and $G(u)$ are smoothed curvature and gradient terms.

3.1. Discretization.

In the following, for an image with resolution $n_x \times n_y$ we take $\Omega = [0,1] \times [0, n_y/n_x]$, such that each pixel represents the value over an $h \times h$ area, where $h = 1/n_x$. Thus, with a discrete input image $\mathbf{g}$ indexed as g_{ij} at points $\mathbf{x}_{ij} = (ih, jh)$ we must discretize (3.1) as

$$V(\mathbf{u}) = d(K\mathbf{u}, \mathbf{g}) + \alpha J(\mathbf{u}), \tag{3.3}$$

where $K : \mathbb{R}^n \to \mathbb{R}^n$ is a discretization of K_c, $\mathbf{u}$ is the output image indexed as u_{ij}, and d and J are discretizations of d_c and J_c. If d_c is an L^p norm, with K_c a restriction to $\Omega \backslash D$, we discretize it as

$$\left(\int_{\Omega \backslash D} |K_c u - g|^p \mathrm{d}\mathbf{x} \right)^{1/p} \approx \left(h^2 \sum_{(i,j)\in\Omega\backslash D} |(K\mathbf{u})_{ij} - g_{ij}|^p \right)^{1/p} =: d(K\mathbf{u}, \mathbf{g}).$$

If J_c is on integral form, one can use quadrature to discretize it as

$$J_c(u) = \int_\Omega H(u)\mathrm{d}\mathbf{x} \approx h^2 \sum_{i,j} H(u)\big|_{\mathbf{x}_{ij}}.$$

Since it requires derivatives of u, $H(u) = C(u)G(u)$ in (3.2) must be approximated at the points $\mathbf{x}_{ij}$ by values $H_{ij}(\mathbf{u})$, such that the final discretization becomes

$$J_c(u) \approx h^2 \sum_{i,j} H_{ij}(\mathbf{u}).$$

For this approximation we use finite differences on a staggered grid as in [30] and [32]. The stencil used for discretizing both $G(u)$ and $C(u)$ is shown in Figure 3.1. The u_{ij} are shown as green squares, and u_x and u_y are approximated by finite differences at red and blue points. With these, we approximate $G(u)$ and $C(u)$. Following the standard approach for TV$_\epsilon$ regularization, $G(u)$ is approximated by backward differences. Approximating $C(u)$ requires evaluation of the x and y components of $\frac{\nabla u}{|\nabla u|_\epsilon}$ at the large dots (red for the x component, blue for the y component) and taking central differences of these to approximate the divergence. To evaluate $|\nabla u|_\epsilon$, we approximate values for u_y at the large red dots and u_x at the large blue dots by the mean of the u_y and u_x approximations at the four nearest blue and red points, respectively. In total, the discretized regularizer is

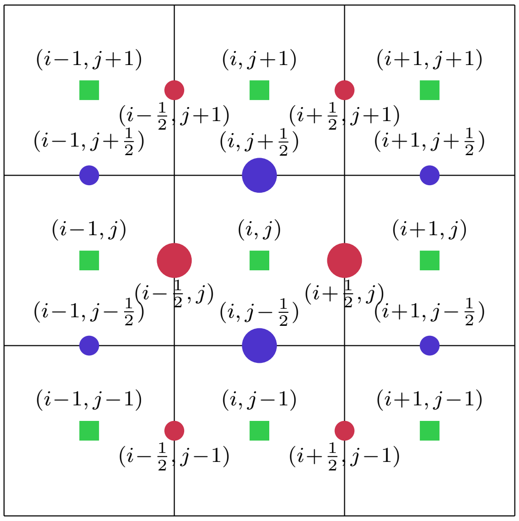

Figure 3.1. *Discretization stencil. Green squares: Pixel data u_{ij}. All red/blue circles: approximations of u_x and u_y. Large red/blue circles: approximations of x and y components of $\nabla u/|\nabla u|_\epsilon$.*

$$J(\mathbf{u}) = \sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\left(a + b\left(\delta_x^+ \frac{\delta_x^- u_{ij}}{w_{i-\frac{1}{2},j}} + \delta_y^+ \frac{\delta_y^- u_{ij}}{w_{i,j-\frac{1}{2}}}\right)^2\right) G_{ij}. \tag{3.4}$$

Here, δ_x^+, δ_x^-, δ_y^+, and δ_y^- denote forward/backward differences in the x and y directions,

$$\delta_x^+ f_{ij} = (f_{i+1,j} - f_{ij})/h, \qquad \delta_x^- f_{ij} = (f_{ij} - f_{i-1,j})/h,$$
$$\delta_y^+ f_{ij} = (f_{i,j+1} - f_{ij})/h, \qquad \delta_y^- f_{ij} = (f_{ij} - f_{i,j-1})/h,$$

and the discretization of the $|\nabla u|_\epsilon$ terms depends on the point as

$$G_{ij} = \sqrt{(\delta_x^- u_{ij})^2 + (\delta_y^- u_{ij})^2 + \epsilon},$$
$$w_{i-\frac{1}{2},j} = \sqrt{(\delta_x^- u_{ij})^2 + (\delta_y^* u_{ij})^2 + \epsilon},$$
$$w_{i,j-\frac{1}{2}} = \sqrt{(\delta_x^* u_{ij})^2 + (\delta_y^- u_{ij})^2 + \epsilon},$$

where

$$\delta_x^* u_{ij} = \frac{1}{4}(\delta_x^- u_{i+1,j} + \delta_x^- u_{ij} + \delta_x^- u_{i+1,j-1} + \delta_x^- u_{i,j-1}),$$
$$\delta_y^* u_{ij} = \frac{1}{4}(\delta_y^- u_{i,j+1} + \delta_y^- u_{ij} + \delta_y^- u_{i-1,j} + \delta_y^- u_{i-1,j+1}).$$

The discrete energy (3.4) has a Lipschitz continuous gradient, where the Lipschitz constant depends on ϵ. Thus, any energy of the form (3.3) using (3.4) as a regularizer will satisfy Theorem 2.3 when the fidelity term has a Lipschitz continuous gradient.

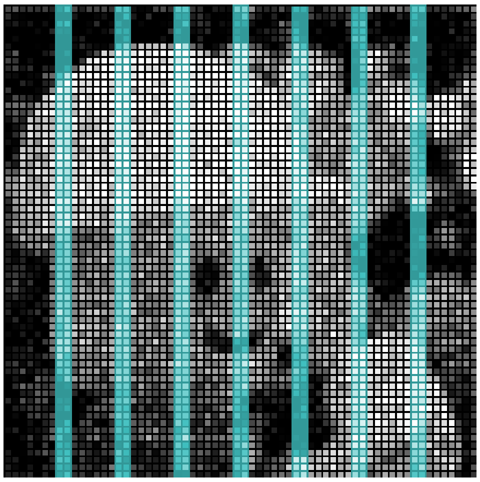

Figure 3.2. *Image split into $M = 8$ and $N = 7$ index sets $\{B_m\}_{m=1}^{8}$ and $\{\Gamma_l\}_{l=1}^{7}$ (marked cyan).*

3.2. Decoupling and parallelization.

Algorithms 2.1 and 2.2 follow a cyclic ordering with elements updated columnwise, but Theorems 2.3 to 2.5 make no assumptions on element ordering, so the convergence rates are unaffected by reordering updates. Also, Figure 3.1 indicates that updating u_{ij} will only affect the $H_{\mu\nu}(\mathbf{u})$ with (μ, ν) immediately surrounding (i, j). Hence, if we split the image in $M + N$ parts as shown in Figure 3.2 and sweep through the cyan elements first, the blocks separated by cyan pixels can be updated independently of each other and thus in parallel, a domain decomposition strategy similar to that in [35]. This inspires Algorithm 3.1, a parallel version of Algorithm 2.1 where the indices of the unknowns are divided into two collections of index sets, $\{B_m\}_{m=1}^{M}$ and $\{\Gamma_l\}_{l=1}^{N}$, based on the dependency radius of V, defined below. Note that the acceleration procedure proposed in Algorithm 2.2 still works here. For a rigorous discussion, we first introduce a distance measure between index sets. Define the distance between two index pairs (i, j) and (k, l) by

$$\mathrm{dist}_{\mathrm{ind}}\left((i,j),(k,l)\right) = \max\left\{|i-k|, |j-l|\right\},$$

which is the graph distance when every index has edges to the closest indices vertically, horizontally, and diagonally. We define the distance between two index sets as

$$\mathrm{dist}_{\mathrm{set}}(I_m, I_n) = \min_{\substack{(i,j)\in I_m\\ (k,l)\in I_n}} \mathrm{dist}_{\mathrm{ind}}\left((i,j),(k,l)\right).$$

If $\mathrm{dist}_{\mathrm{set}}(I_m, I_n) = 0$, then I_m and I_n share at least one index; if $\mathrm{dist}_{\mathrm{set}}(I_m, I_n) = 1$, at least one index in I_m is adjacent to an index in I_n, horizontally, vertically, or diagonally; if $\mathrm{dist}_{\mathrm{set}}(I_m, I_n) = 2$, there is a band of width 1 of indices separating I_m and I_n, et cetera. We can now define the dependency radius of a function; we say that $V : \mathbb{R}^n \to \mathbb{R}$ has dependency radius R if for all $\delta \in \mathbb{R}$ and (i, j),

$$V(\mathbf{u} + (\delta - u_{ij})\mathbf{e}_{ij}) - V(\mathbf{u}) = F(\mathbf{u}_{ij}^{R}(\mathbf{u}), \delta),$$

Algorithm 3.1 DG-PARALLEL

```
Choose $\tau > 0$, $tol > 0$ and $\mathbf{u}^0 \in \mathbb{R}^n$. Set $k = 0$. Initialize $M$ threads.
Define $M$ index sets $B_m$ and $N$ index sets $\Gamma_l$.
repeat
  parallel : $N$ threads. Thread number $l$ does:
  $\mathbf{v}_0^{k,l} = \mathbf{u}^k$
  for $j \in \Gamma_l$ do
    Solve $\beta_j^k = -\tau(V(\mathbf{v}_{j-1}^{k,l} + \beta_j^k \mathbf{e}_j) - V(\mathbf{v}_{j-1}^{k,l}))/\beta_j^k$
    $\mathbf{v}_j^{k,l} = \mathbf{v}_{j-1}^{k,l} + \beta_j^k \mathbf{e}_j$
  end for
  Reduce: $\mathbf{v}_0^k = \mathbf{u}^k + \sum_{j \in \cup_{l=1}^N \Gamma_l} \beta_j^k \mathbf{e}_j$
  Parallel : $M$ threads. Thread number $m$ does:
  $\mathbf{v}_0^{k,m} = \mathbf{v}_0^k$
  for $j \in B_m$ do
    Solve $\beta_j^k = -\tau(V(\mathbf{v}_{j-1}^{k,m} + \beta_j^k \mathbf{e}_j) - V(\mathbf{v}_{j-1}^{k,m}))/\beta_j^k$
    $\mathbf{v}_j^{k,m} = \mathbf{v}_{j-1}^{k,m} + \beta_j^k \mathbf{e}_j$
  end for
  Reduce: $\mathbf{u}^{k+1} = \mathbf{v}_0^k + \sum_{j \in \cup_{m=1}^M B_m} \beta_j^k \mathbf{e}_j$
  $k = k + 1$
until $\left(V(\mathbf{u}^k) - V(\mathbf{u}^{k-1})\right) / V(\mathbf{u}^0) < tol$
```

where $F : \mathbb{R}^{(2R+1)^2+1} \to \mathbb{R}$ is a function depending on δ and

$$\mathbf{u}_{ij}^R(\mathbf{u}) = (u_{i_{\min}, j_{\min}}, \dots, u_{ij}, \dots, u_{i_{\max}, j_{\max}}),$$

where

$$\begin{aligned} i_{\min} &= \max\{i - R, 1\}, & j_{\min} &= \max\{j - R, 1\}, \\ i_{\max} &= \min\{i + R, n_x\}, & j_{\max} &= \min\{j + R, n_y\}. \end{aligned}$$

This means that computing the change in V from updating unknown number (i, j) requires only the unknowns with indices within a distance of R. In the discretized Euler's elastica problem we have $R = 1$. If V has dependence radius R, then u_{ij} can be updated using the Itoh–Abe discrete gradient independently of u_{kl} if $\mathrm{dist}_{\mathrm{ind}}((i,j),(k,l)) > R$. We can decouple a problem with dependency radius R by choosing M index sets $B_m \subset \Omega$ such that $\mathrm{dist}_{\mathrm{set}}(B_m, B_n) > R$ for all (m, n). Then, one chooses a second collection of N index sets Γ_l such that $\mathrm{dist}_{\mathrm{set}}(\Gamma_k, \Gamma_l) > R$ and $\cup_{l=1}^N \Gamma_l = \Omega / \cup_{m=1}^M B_m$. Note that, in general, $M \neq N$ and that while this discussion has been focused on two-dimensional indexing, generalizing $\mathrm{dist}_{\mathrm{ind}}$ and $\mathrm{dist}_{\mathrm{set}}$ in the obvious manner to higher-dimensional index pairs admits a similar approach in arbitrary indexing dimensions.

3.3. Effect of dependency radius on complexity of the algorithm.

A consequence of V having dependency radius R is that $\partial V / \partial u_{ij}$ depends on $\mathbf{u}_{ij}^R$ only. This can be used to obtain sharper versions of Theorems 2.3 to 2.5 through a property presented in the following lemma for two-dimensional indexing.

Lemma 3.1. *If Assumption 2.2 holds and in addition V has dependency radius R, the $\mathbf{u}^k$ produced by Algorithm 2.2 satisfy*

$$\|\nabla V(\mathbf{u}^k)\|^2 \leq \nu \left(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1})\right) \tag{3.5}$$

with

$$\nu = 2L_{\max}^2 \left((2R+1)^2 \tau_{\max} + \frac{\tau_{\max}}{L_{\max}^2 \tau_{\min}^2}\right).$$

Proof. Recall the coordinatewise formulation of the Itoh–Abe scheme, with two-dimensional indexing where, with $\beta_{lm}^k = u_{lm}^{k+1} - u_{lm}^k$,

$$\beta_{lm}^k = -\tau_k \frac{V\left(\mathbf{v}_{l,m}^k\right) - V\left(\mathbf{v}_{l,m-1}^k\right)}{\beta_{lm}^k}.$$

Here, $\mathbf{v}_{l,m}^k := \mathbf{u}^k + \sum_{i=1}^{l-1} \sum_{j=1}^{n_y} \beta_{ij}^k \mathbf{e}_{ij} + \sum_{j=1}^{m} \beta_{lj}^k \mathbf{e}_{lj}$. We follow the proof of Theorem 2.3 up to (2.7), where we exploit the dependence radius R of V. For an $s \in (0,1)$, we have

$$\begin{aligned}\left|\frac{\partial V}{\partial u_{lm}}(\mathbf{u}^{k+1})\right| &= \left|\frac{\partial V}{\partial u_{lm}}(\mathbf{u}_{lm}^R(\mathbf{u}^{k+1}))\right| \\ &\leq \left|\frac{\partial V}{\partial u_{lm}}(\mathbf{u}_{lm}^R(\mathbf{u}^{k+1})) - \frac{\partial V}{\partial u_{lm}}(\mathbf{u}_{lm}^R(\mathbf{v}_{l,m-1}^k) + s\beta_{lm}^k \mathbf{e}_{lm})\right| + \frac{1}{\tau_k}\left|\beta_{lm}^k\right|.\end{aligned}$$

Using coordinatewise Lipschitz continuity, we have

$$\left|\frac{\partial V}{\partial u_{lm}}(\mathbf{u}^{k+1})\right| \leq L_{lm}\|\mathbf{u}_{lm}^R(\mathbf{u}^{k+1}) - \mathbf{u}_{lm}^R(\mathbf{v}_{l,m-1}^k) - s\beta_{lm}^k \mathbf{e}_{lm}\| + \frac{1}{\tau_k}\left|\beta_{lm}^k\right|,$$

and summing up over all coordinates, we get

$$\begin{aligned}\|\nabla V(\mathbf{u}^{k+1})\|^2 &\leq \sum_{l=1}^{n_x}\sum_{m=1}^{n_y}\left(L_{lm}\|\mathbf{u}_{lm}^R(\mathbf{u}^{k+1}) - \mathbf{u}_{lm}^R(\mathbf{v}_{l,m-1}^k) - s\beta_{lm}^k \mathbf{e}_{lm}\| + \frac{1}{\tau_k}\left|\beta_{lm}^k\right|\right)^2 \\ &\leq 2\sum_{l=1}^{n_x}\sum_{m=1}^{n_y}\left(L_{lm}^2\left(\sum_{i=l_{\min}}^{l_{\max}}\sum_{j=m_{\min}}^{m_{\max}} {\beta_{ij}^k}^2\right) + \frac{1}{\tau_k^2}\left|\beta_{lm}^k\right|^2\right) \\ &\leq 2L_{\max}^2\left((2R+1)^2 + \frac{1}{L_{\max}^2 \tau_{\min}^2}\right)\|\mathbf{u}^{k+1} - \mathbf{u}^k\|^2.\end{aligned}$$

Since $\|\mathbf{u}^{k+1} - \mathbf{u}^k\|^2 = \tau_k(V(\mathbf{u}^k) - V(\mathbf{u}^{k+1}))$, this concludes the proof. ■

Remark. This improved estimate affects the complexity of Theorems 2.3 to 2.5, reducing it from a worst-case of $\mathcal{O}(n^{1/2})$ to $\mathcal{O}(R)$ for convex two-dimensionally indexed problems with optimal time steps. Indeed, choosing a constant step size τ minimizing ν, one obtains

$$\tau = \frac{1}{(2R+1)L_{\max}}, \qquad \nu = 4(2R+1)L_{\max}.$$

Note that for problems involving discretizations such as the Euler's elastica regularization considered here, $L_{\max}$ may still depend on n. Regardless, this is an improvement over the $\mathcal{O}(nL_{\max})$ or worse complexity of other coordinate descent methods detailed in [34]. For general D-dimensional indexing one can expect the complexity to scale as $\mathcal{O}(R^{D/2})$ with optimal step size selection.

4. Numerical experiments.

In this section, we first apply Euler's elastica regularization to denoising problems and to image inpainting. The results are compared to those of TV_ϵ regularization to verify the qualitative improvement of Euler's elastica regularization with Algorithm 2.1 over TV. This is not intended as an account on the competitiveness of Euler's elastica against other regularizing procedures in general but serves as a proof of concept for the qualitative and algorithmic performance of the discrete gradient approach for Euler elastica when compared to discrete gradient schemes for TV. We further investigate the convergence rate numerically, with varying smoothing constant ϵ and ordering of the unknowns. We also verify the predictions of Lemma 3.1. Next, we compare the execution time to another state-of-the-art algorithm for nonconvex optimization, the iPiano algorithm [24], and to the gradient descent and heavy-ball algorithms. Finally, we evaluate the algorithm's sensitivity to the initial guess.

All algorithms were implemented as hybrid MATLAB and C functions using the MATLAB EXecutable (MEX) interface, where critical parts of the code are implemented in C. The tests were executed using MATLAB (2017a release) running on a mid-2014 MacBook Pro with a four-core 2.5 GHz Intel Core i7 processor and 16 GB of 1600 MHz DDR3 RAM. For the Brent–Dekker algorithm implementation we used the built-in MATLAB function `fzero`, and block parallelization was done using MATLAB's `blockproc` and `parfor` functions.

4.1. Image denoising.

We first consider denoising images. The typical choice of fidelity term is an L^p metric where p depends on the type of noise encountered. The discretized forward operator K is the identity operator. We wish to minimize

$$V(\mathbf{u}) = \sum_{i,j} |u_{ij} - g_{ij}|^p + J(\mathbf{u}). \tag{4.1}$$

In the first example we have added Gaussian noise with a standard deviation of 0.2, using $p = 2$ for the fidelity term. In the second example we have added impulse noise, randomly setting the values of 25% of the pixels to either 0 or 1 as depicted in the top pictures in Figure 4.1. Impulse noise is removed using $p = 1$ in (4.1). Note that the fidelity term with $p = 1$ is nondifferentiable and falls outside of the theoretical basis of subsection 2.2. However, results in [27] show that the Itoh–Abe scheme will converge to a stationary point when using $p = 1$ in (4.1). Therefore, this example is included to see how the method behaves beyond the smooth setting.

Figure 4.2 shows Euler's elastica denoising with $p = 2$ applied to an image corrupted by Gaussian noise in its lower right-hand panel and a TV_ϵ regularized version in the lower left-hand panel. For both TV_ϵ and elastica denoising, we chose $\epsilon = 10^{-4}$; larger values resulted

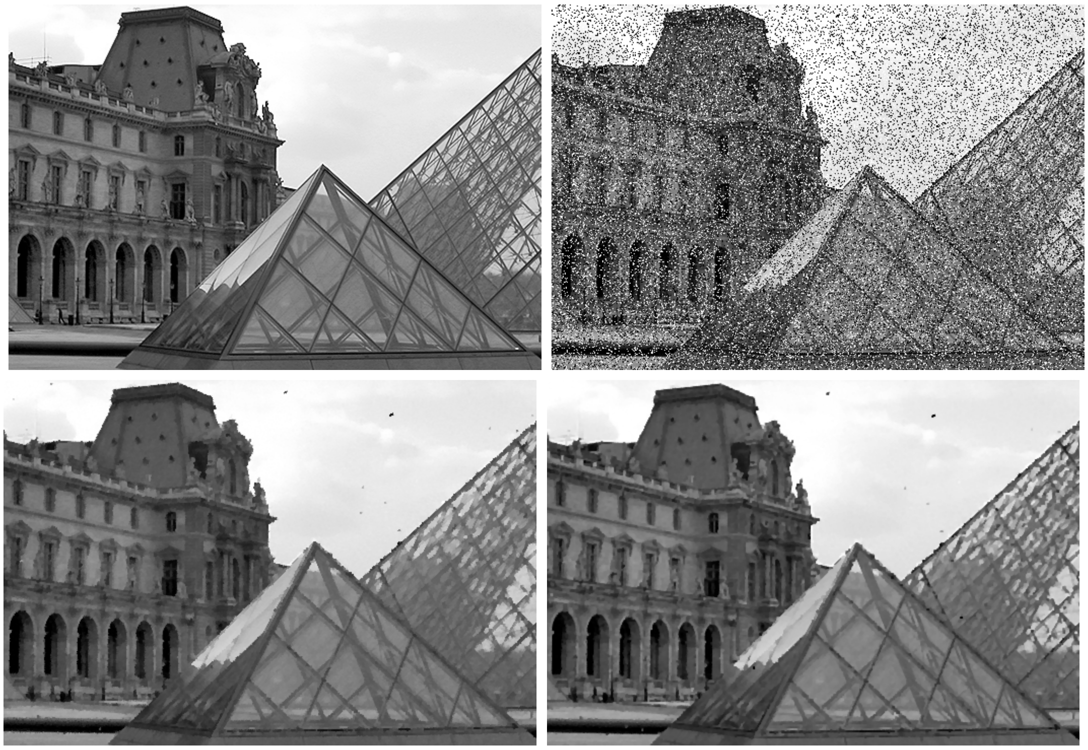

Figure 4.1. *Denoising with* $p = 1$*. Left:* TV_ϵ *denoised. PSNR:* 25.7287*; SSIM:* 0.8572*. Right: Elastica denoised. PSNR:* 25.9611*; SSIM:* 0.8628*.*

in blurring and lower values showed no visible improvement but slower convergence. For TV_ϵ denoising, we set $a = 0.17$, and for elastica denoising, we chose $a = 0.9$ and $b = 0.9$. These values were chosen to maximize peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM).

In the lower right-hand panel of Figure 4.1, the result of elastica denoising with $p = 1$ on a picture of the Louvre, corrupted by impulse noise, is shown. A TV_ϵ regularized version is seen in the lower left-hand panel. As above, we chose $\epsilon = 10^{-4}$ for both TV_ϵ and elastica denoising. In the TV_ϵ case, we set $a = 0.8$, and in the elastica case, we chose $a = 0.4$ and $b = 0.2$ to maximize PSNR and SSIM.

As can be seen in both examples, the results of Euler's elastica denoising are slightly more visually appealing than the TV_ϵ denoised versions, which is as expected since Euler's elastica generalizes TV regularization. In Figure 4.2, edges are sharper and the contrast level better in the elastica denoised image. In Figure 4.1 the pyramids' lines are sharper and the museum's façade details are clearer. One can also see that the textures are smoother and that edges are less jagged in the elastica reconstruction.

4.2. Image inpainting.

Inpainting is used when there is data loss in known pixels. Using $p = 2$ and a discretized restriction operator K we wish to minimize

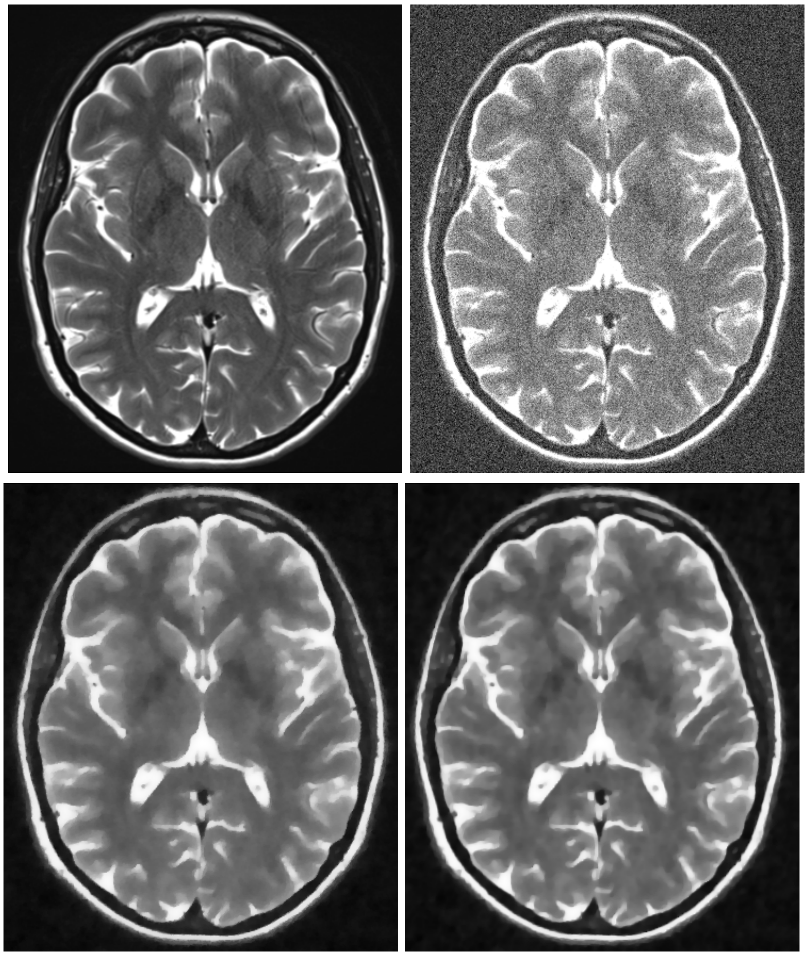

Figure 4.2. *Denoising with* $p = 2$. *Top left: original image. Top right: noisy input image* g. *Bottom left:* TV_ϵ *denoised. PSNR:* 20.9421*; SSIM:* 0.8398. *Bottom right: Elastica denoised. PSNR:* 21.3760*; SSIM:* 0.8595.

$$V(\mathbf{u}) = \sum_{(i,j)\in\Omega\setminus D} (u_{ij} - g_{ij})^2 + J(\mathbf{u}),$$

where $D \subset \Omega$ is the damaged domain. Figure 4.3 shows the result of applying Algorithm 2.1 to an example inpainting problem. Here, the top left panel shows the original image, the top right panel the image with 95% of the pixels removed randomly, the bottom left panel a TV_ϵ inpainted image, and the bottom right panel an Euler's elastica inpainted image. For both TV_ϵ and elastica, we chose $\epsilon = 10^{-4}$. In the TV_ϵ case, we set $a = 2.5 \cdot 10^{-7}$, and in the elastica case, we chose $a = 10^{-6}$ and $b = 10^{-5}$. These values were chosen to maximize SSIM. Here, the superiority of Euler's elastica is evident, as more details are reconstructed and the image appears sharper than with TV_ϵ regularization.

4.3. Convergence rates.

When denoising using $p = 2$ in (4.1), the conditions of Theorem 2.3 are fulfilled, and so we investigate the convergence rates numerically in this case. The

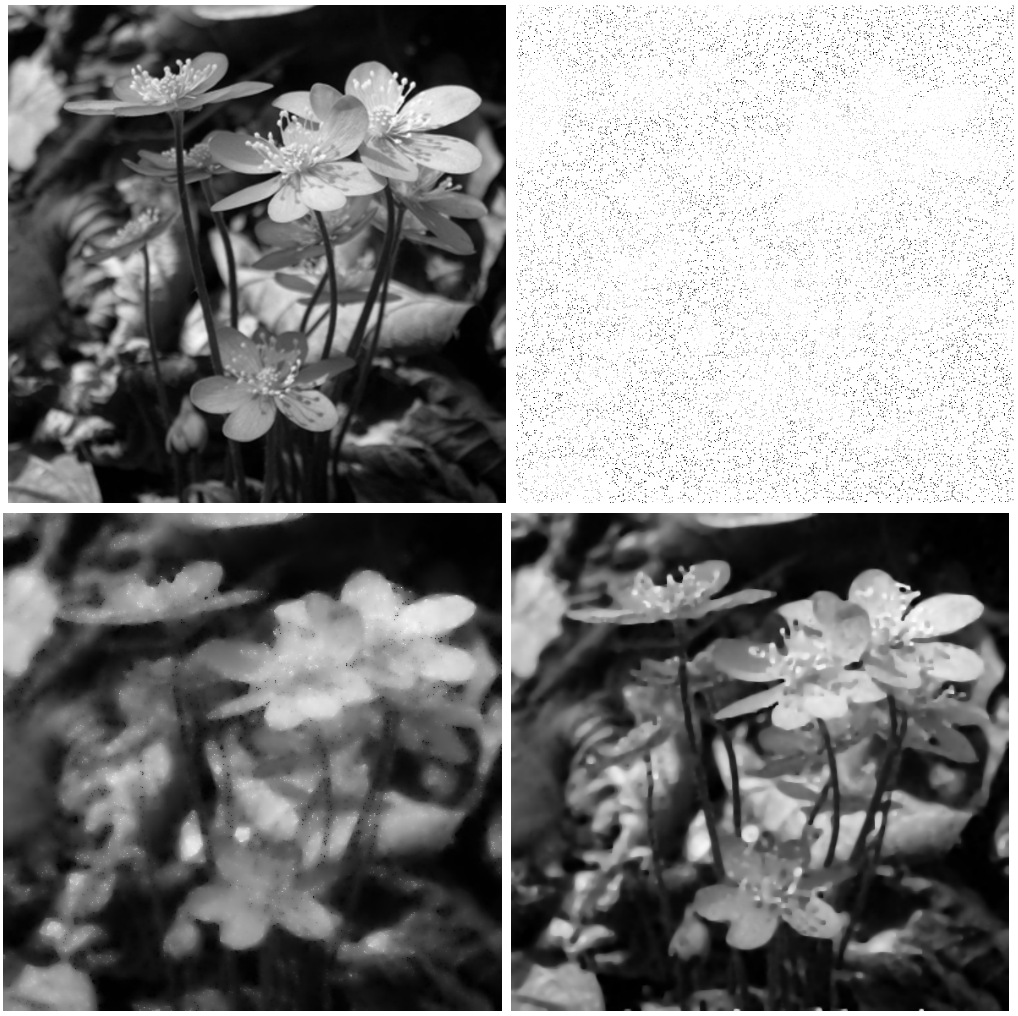

Figure 4.3. *Top left: Original image. Top right:* 95% *random data loss. Bottom left: Inpainted with* TV_ϵ, *SSIM:* 0.7512. *Bottom right: Inpainted with elastica, SSIM:* 0.8896.

two plots in Figure 4.4 show $\min_{0\le l\le k}\|\nabla V(\mathbf{u}^l)\|/\|\nabla V(\mathbf{u}^0)\|$ for DG and DG-ADAPT applied to the Euler's elastica regularized denoising problem with $p=2$ shown in Figure 4.2 for two choices of ϵ. Each plot shows the result of initializing with a τ_0 chosen by trial and error to yield the best convergence rate, and with a much smaller τ_0. Both algorithms were started from the same random initalization $\mathbf{u}^0$ and with the same initial time step τ_0. For DG-ADAPT, the additional parameters were chosen as $\rho = 0.99$, $c_1 = 0.7, c_2 = 0.9$, and $\gamma = 1.005$. Note that the left-hand plot is semilogarithmic, while the right-hand plot is logarithmic. The left-hand plot shows linear convergence for the DG algorithm when τ_0 is chosen correctly and a much slower rate for the suboptimal τ_0. The linear convergence can be expected by Theorem 2.5 if the choice of $\epsilon = 10^{-4}$ means V, which is twice differentiable, becomes strongly convex in a neighborhood of the minimizer, or if it is a PŁ-function. In the right-hand plot, where $\epsilon = 10^{-7}$ leads to a more ill-conditioned problem, the convergence rate appears closer to that predicted

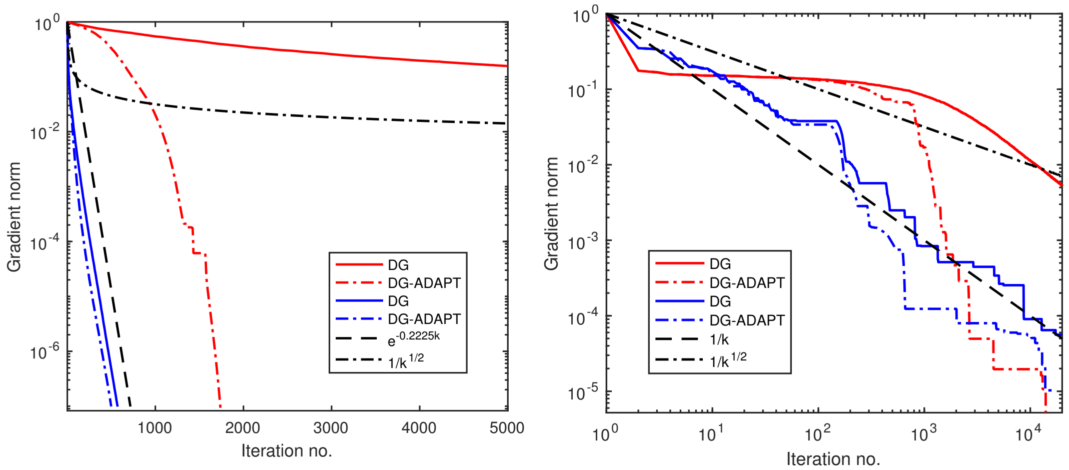

Figure 4.4. *Convergence rates in terms of* $\min_{0\le l\le k}\|\nabla V(\mathbf{u}^l)\|/\|\nabla V(\mathbf{u}^0)\|$ *for the Euler's elastica regularized denoising problem with* $p=2$ *illustrated in Figure* 4.2*. Blue denotes* $\tau_0 = 0.38$*, red denotes* $\tau_0 = 0.38\cdot 10^{-4}$*. Left: Using* $\epsilon = 10^{-4}$*. Right: Using* $\epsilon = 10^{-7}$*.*

in Theorem 2.4, indicating that a neighborhood of strong convexity has not yet been reached. Both plots show that using adaptive step sizes yields faster convergence than fixed step sizes, especially when τ_0 is not carefully chosen beforehand.

Figure 4.5 shows convergence rates for four different element orderings. The first ordering is the natural ordering which iterates over pixels starting in one corner and proceeding columnwise. The second is red-black ordering where pixels u_{ij} with $i+j$ even are updated first, then pixels with $i+j$ odd. Third is a random ordering, with the same ordering used for all time steps. Last, we consider the block ordering of the parallelized algorithm as illustrated in Figure 3.2. The plots of Figure 4.5 concern the same problem as Figure 4.4, but with the DG algorithm only and showing rates in terms of the relative optimality error $(V(\mathbf{u}^k)-V^*)/(V(u^0)-V^*)$, where V^* was produced by running the algorithm for 20,000 iterations. Step sizes were chosen individually for the different orderings to produce the best possible convergence. The left-hand plot plateaus at a relative optimality error of $\sim 10^{-16}$, i.e., machine precision. Both plots show that the asymptotic convergence rate, represented by the slope of the lines, is similar for all orderings, as is to be expected from the independency of ordering in the convergence theorems. However, the early rate of the random and red-black orderings of the left-hand plot is better than that of the natural and block orderings, suggesting that the choice of ordering affects the constant ν in Theorems 2.3 to 2.5. We consider this an interesting topic for further investigation.

Figure 4.6 shows convergence rates in terms of $(V(\mathbf{u}^k)-V^*)/(V(\mathbf{u}^0)-V^*)$ for a TV$_\epsilon$ regularized inpainting problem where a square of width $1/2$ is missing from the middle of an all-black square image, i.e., $\Omega=[0,1]^2$, $D=[1/4,3/4]^2$ and $g=0$ on $\Omega\backslash D$. With the parameter choices $a=1/16$ and $\epsilon=0.1$, we test with image resolutions $2^m\times 2^m$, $m=5,6,7,8,9$ to verify the conclusion of Lemma 3.1. It can be shown that the objective function of the TV$_\epsilon$ regularized inpainting problem satisfies the assumptions of Theorem 2.5 with $\sigma=h^2$ and $L_{\max}=h^2(1+4a\epsilon^{-1/2}h^{-2})$. Hence, with ν chosen optimally as explained in the remark after

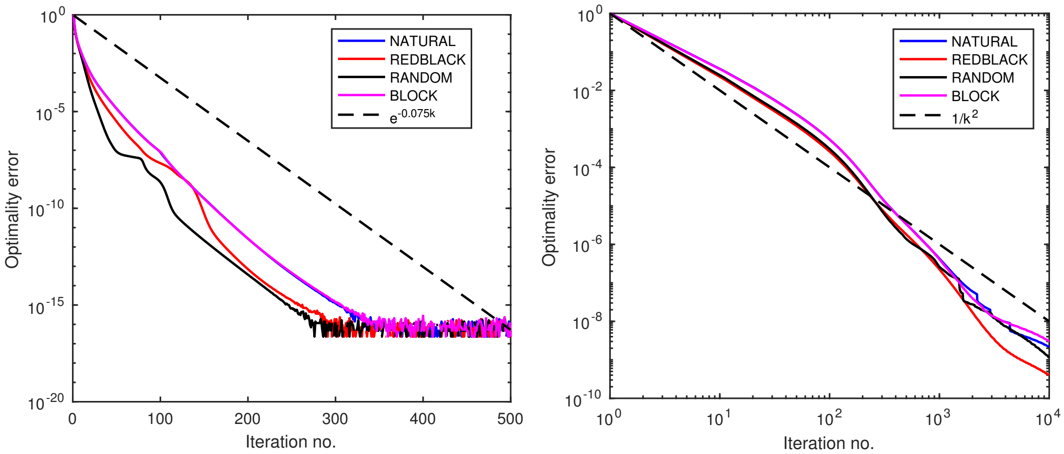

Figure 4.5. *Convergence rates in terms of* $(V(\mathbf{u}^k) - V^*)/(V(\mathbf{u}^0) - V^*)$ *for the Euler's elastica regularized denoising problem with* $p = 2$ *illustrated in Figure 4.2, with different orderings. Left: Using* $\epsilon = 10^{-4}$*. Right: Using* $\epsilon = 10^{-7}$*.*

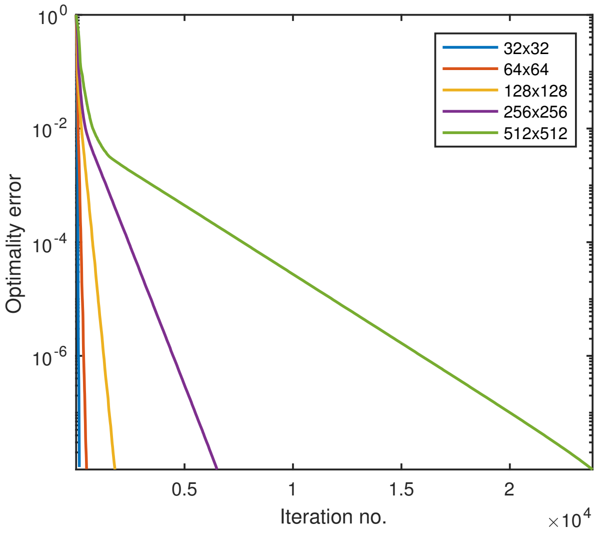

Figure 4.6. *Convergence rates in terms of* $(V(\mathbf{u}^k) - V^*)/(V(\mathbf{u}^0) - V^*)$ *for the* TV_ϵ *square inpainting problem with* $p = 2$*, for varying problem sizes.*

Lemma 3.1 one should expect the following rate of convergence for the logarithmic error:

$$\log\left(\frac{V(\mathbf{u}^k) - V^*}{V(\mathbf{u}^0) - V^*}\right) \leq k \log\left(1 - \frac{2\sigma}{\nu}\right) \approx k\frac{2(2R+1)\epsilon^{1/2}}{a}h^2.$$

This is observed in Figure 4.6. For each resolution, the iterations eventually converge at a fixed linear rate. This rate, obtained by exponential fitting, decreases by a factor of 4 as $n = 2^{2m}$ quadruples.

4.4. Execution time. As a general algorithm suited to the kind of nonconvex minimization problems that the Euler's elastica problem poses, it is reasonable to compare the DG algorithms to the iPiano algorithm of [24]; in particular, we use Algorithm 4 from this article. Inspired by Polyak's heavy-ball algorithm and the proximal gradient algorithm, iPiano considers minimization problems of the form

$$\min_{\mathbf{u}\in\mathbb{R}^n} f(\mathbf{u}) + g(\mathbf{u}),$$

where g is convex and possibly nonsmooth while f is smooth and possibly nonconvex, and iterates based on the update scheme

$$\mathbf{u}^{k+1} = (I+\alpha\partial g)^{-1}(\mathbf{u}^k - \alpha\nabla f(\mathbf{u}^k) + \beta(\mathbf{u}^k - \mathbf{u}^{k-1})),$$

where $(I+\alpha\partial g)^{-1}$ denotes a proximal step by

$$(I+\alpha\partial g)^{-1}(\mathbf{y}) = \arg\min_{\mathbf{z}\in\mathbb{R}^n} \frac{\|\mathbf{z}-\mathbf{y}\|^2}{2} + \alpha g(\mathbf{z}).$$

We wish to time the algorithms on a problem with nondifferentiable terms, and so we use a variation on the discrete elastica regularizer (3.4), taking

$$J(\mathbf{u}) = a\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\overline{G}_{ij} + b\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}\left(\delta_x^+\frac{\delta_x^- u_{ij}}{w_{i-\frac{1}{2},j}} + \delta_y^+\frac{\delta_y^- u_{ij}}{w_{i,j-\frac{1}{2}}}\right)^2 G_{ij} =: aT(\mathbf{u}) + bK(\mathbf{u}),$$

where

$$\overline{G}_{ij} = \sqrt{(\delta_x^- u_{ij})^2 + (\delta_y^- u_{ij})^2}.$$

That is, the TV term T is not differentiable but the curvature term K is. The choice of f and g in the iPiano algorithm depends on whether the fidelity term is differentiable ($p=2$) or not ($p=1$). We take $f = K + d$ if $p = 2$, but $f = K$ if $p = 1$. Likewise, we take $g = T$ if $p = 2$, but $g = T + d$ if $p = 1$. In both cases, evaluating $(I+\alpha\partial g)^{-1}$ is equivalent to solving a TV regularization problem, which is done efficiently using the Chambolle–Pock algorithm [8]. This algorithm can be accelerated in the case of a uniformly convex fidelity term, i.e., if d is a discrete L^2 norm. If it is not, as is the case when d is a discrete L^1 norm, no acceleration is possible. The DG-ADAPT algorithm requires the computation of gradients; they are computed using the smoothed (3.4); also, when $p = 1$, the additional smoothing

$$\|u-g\|_1 = \sum_{i,j}|u_{ij} - g_{ij}| \approx \sum_{i,j}\sqrt{(u_{ij}-g_{ij})^2 + \varepsilon} := \|u-g\|_{1,\varepsilon}$$

was used with $\varepsilon = 10^{-12}$ to compute gradients in the DG-ADAPT algorithm.

All algorithms were implemented in serial versions using MEX and were timed. Note that the above nonsmoothed TV term was used in the DG/DG-ADAPT algorithms as well for this test. Tables 4.1 and 4.2 show timing results for a denoising test on a 512×512 image for different values of ϵ using a discrete L^2 norm and a discrete L^1 norm for the fidelity term, respectively. For each ϵ, a reference solution $\bar{V}$ was found by running the DG algorithm for 20,000 iterations or until a minimizer was found with machine precision. The algorithms were

Table 4.1

Results of L^2 test. Format: (Iterations/CPU time (s)). Best times in bold.

ϵ	iPiano	DG	DG-ADAPT
10^{-1}	**30/8.90**	29/15.61	28/17.39
10^{-2}	**32/9.60**	28/14.79	27/16.54
10^{-3}	**38/12.40**	38/20.42	35/21.67
10^{-4}	**59/16.78**	67/38.22	57/36.62
10^{-5}	**216/47.74**	115/69.51	89/60.20
10^{-6}	1684/556.11	180/115.26	**131/91.19**
10^{-7}	3968/1071.37	269/196.12	**204/151.71**

Table 4.2

Results of L^1 test. Format: (Iterations/CPU time (s)). Best times in bold.

ϵ	iPiano	DG	DG-ADAPT
10^{-1}	**23/21.73**	199/171.82	168/161.73
10^{-2}	**59/29.67**	288/250.23	247/231.56
10^{-3}	**146/46.36**	305/255.12	252/231.50
10^{-4}	**401/229.04**	300/246.27	253/223.21
10^{-5}	1399/2181.81	**303/246.90**	292/257.47
10^{-6}	4000/18566.24	**303/242.00**	304/267.11
10^{-7}	N/A	**400/335.10**	423/373.91

then tested on the problem, running until the iterations reached a value of $V(\mathbf{u}^k) \leq 1.0001 \cdot \bar{V}$ or 4000 iterations. Both algorithms require the solution of a subproblem; a root finding problem for the DG algorithms and the evaluation of a proximal operator for the iPiano algorithm. For the DG algorithms the tolerance of the root finding algorithm was kept at a fixed value, while for the iPiano algorithm, the tolerance in the prox operator evaluation was adjusted to obtain the fastest runtime while still converging. In DG-ADAPT, the parameter choices $c_1 = 0.7, c_2 = 0.9, \rho = 0.98$, and $\gamma = 1.005$ were used for the L^2 test, and $c_1 = 0.2, c_2 = 0.7, \rho = 0.995$, and $\gamma = 1.0025$ were used for the L^1 test.

From both tables, it is apparent that the DG and DG-ADAPT algorithms both scale better with ϵ than iPiano, which reached the maximum number of iterations at $\epsilon = 10^{-6}$ in the L^1 test and hence was not timed with $\epsilon = 10^{-7}$. However, for larger values of ϵ, iPiano appears to be the better choice. The time usage per iteration increases with ϵ for both iPiano and DG/DG-ADAPT; for iPiano, this is due to the precision in the prox operator evaluation increasing which requires more time. For DG/DG-ADAPT, the slight increase can be explained by an increase in the amount of iterations needed by the Brent–Dekker algorithm to solve the scalar subproblems. Also note that in Table 4.2, we can see that DG-ADAPT is not noticeably faster than DG in most cases, indicating that using gradients of smoothed versions of the objective function is not sufficient to accelerate convergence for nonsmooth problems.

Table 4.3 shows timing results of denoising test problems using the smoothed elastica regularizer (3.4), on a 512×512 image using a squared L^2 fidelity term, comparing the DG and DG-ADAPT algorithms to the gradient descent and heavy-ball algorithms with Armijo step size selection. Here, the parameters of the DG-ADAPT algorithm were chosen as $\rho = 0.99$, $c_1 = 0.7$, $c_2 = 0.9$, and $\gamma = 1.005$ after experimentation. The τ parameter in the DG algorithm

Table 4.3

Results of L^2 test with smooth TV_ϵ term. Format: (Iterations/CPU time (s)). Best times in bold.

ϵ	Gradient descent	Heavy-ball	DG	DG-ADAPT
10^{-1}	**12/5.16**	23/12.79	20/9.46	19/8.61
10^{-2}	27/14.78	29/17.95	25/12.72	**24/11.85**
10^{-3}	145/101.00	38/25.16	32/17.43	**30/15.76**
10^{-4}	388/319.57	96/73.59	**42/24.66**	43/24.71
10^{-5}	1310/1271.73	312/284.49	**68/43.95**	76/47.92
10^{-6}	4001/4621.38	1018/1079.10	119/83.15	**112/73.07**
10^{-7}	N/A	2922/3765.52	194/142.05	**181/116.88**
10^{-8}	N/A	4001/5488.83	376/291.73	**398/260.18**

Table 4.4

Results of inpainting test. Format: (Final energy/CPU time (s)). Best times in bold.

Initialization	iPiano	DG	DG-ADAPT
Random	0.012334/3434	**0.012323/431**	0.012323/460
Unicolor	0.014892/2934	0.012324/2740	**0.012324/438**
Original	0.012263/1131	**0.012263/145**	0.012263/208

was chosen to give the fastest convergence at $\epsilon = 10^{-4}$, and the same τ was used as initial τ_0 in DG-ADAPT. The table shows that the DG and DG-ADAPT algorithms outperform the gradient descent and heavy-ball algorithms on the problem for all ϵ except $\epsilon = 10^{-1}$, with increasing difference as $\epsilon \to 0$. The gradient descent algorithm reached the maximum of 4000 iterations at $\epsilon = 10^{-6}$ and was therefore not tested with smaller ϵ.

Using the DG-PARALLEL algorithm with column splitting, a speedup of 2–2.5 was observed when employing four cores, with larger speedup values for larger images.

4.5. Dependence on starting point.

To investigate the influence of the starting point on the performance of the algorithm in the minimization of the nonconvex Euler elastica problem, we tested the inpainting problem with DG and iPiano, comparing execution time and reconstruction quality of the two algorithms when starting from a random or unicolor (black) starting image, or from the original image.

As seen in Figure 4.7, the DG algorithm produces different but acceptable reconstructions depending on the starting guess. The iPiano algorithm works when starting from a random image and the original image, but is worse with a unicolor starting image. Assuming that starting from the original image gives the best reconstruction, we compare reconstructions starting from other images to this. Figure 4.8 shows differences between images obtained starting from the original and the images obtained when starting from random or unicolor images. We can see that the reconstructions are largely in agreement except in certain areas such as the stamens of the top right flower. In Table 4.4 we see that the iPiano algorithm is slower than the DG algorithms when it comes to inpainting and also that the unicolor initialization produced an answer which was pretty far from optimal as compared to the other initializations.

5. Conclusion.

We have introduced a novel method for solving nonconvex optimization problems and tested it on Euler's elastica regularized variational image analysis problems.

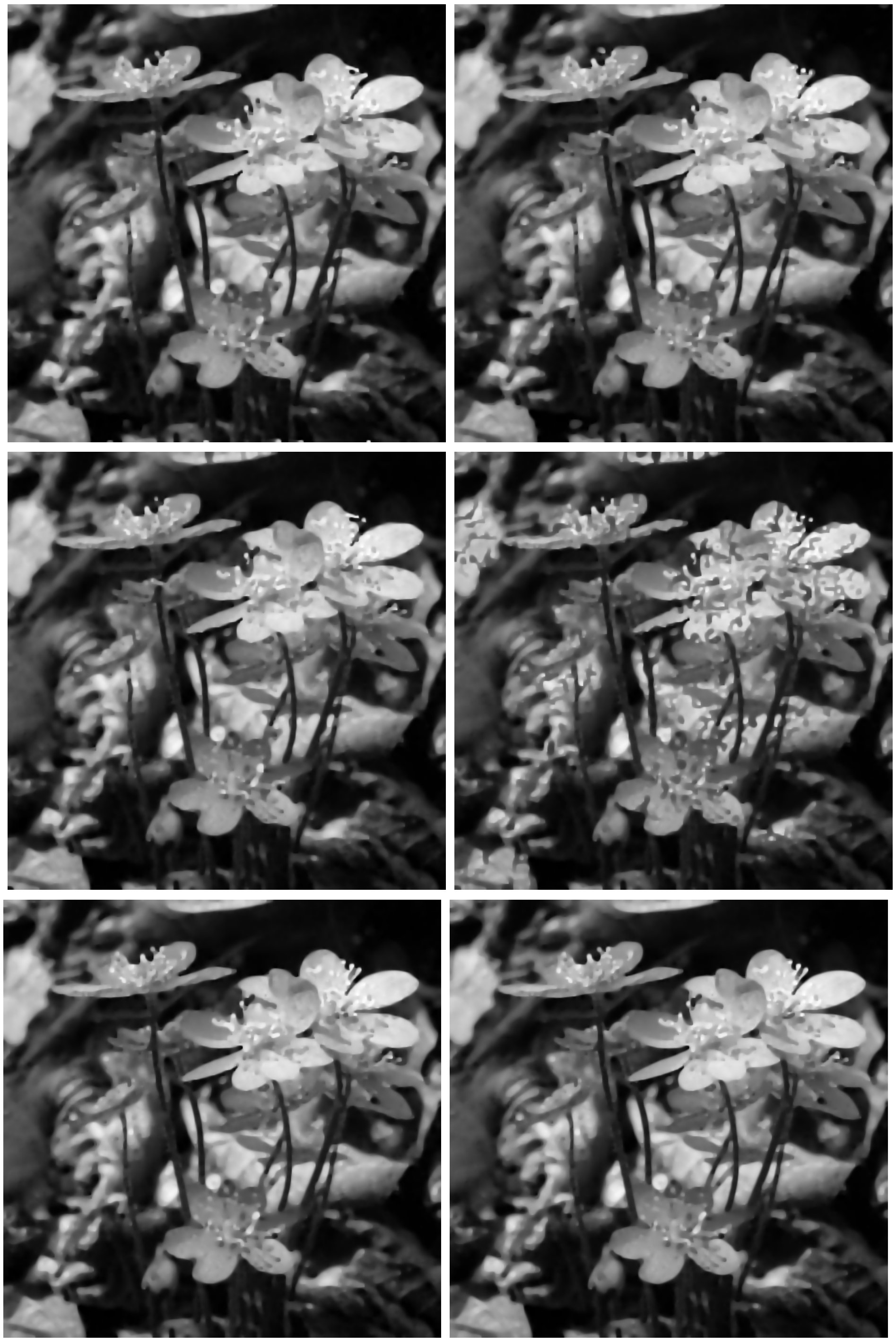

Figure 4.7. *Inpainting with different starting values. Left column: DG. Right column: iPiano. Top: Random initial value. Middle: Unicolor (black) initial value. Bottom: Original initial value.*

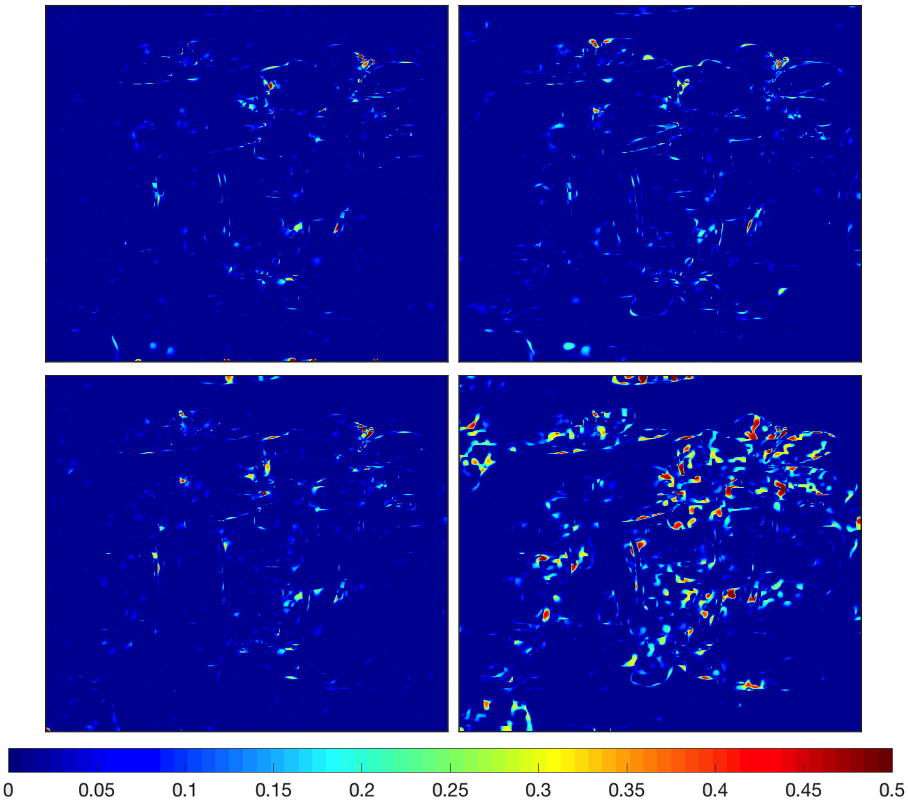

Figure 4.8. *Difference from optimum when inpainting with different starting values. Left column: DG. Right column: iPiano. Top: Random initial value. Bottom: Unicolor nitial value.*

We have produced a convergence rate estimate for nonconvex problems assuming that V is continuously differentiable with Lipschitz continuous gradient. This rate does not depend on the problem size n but rather on the dependency radius R of V. Numerical tests confirm the quality of images denoised and inpainted with Euler's elastica as a regularizer, that the time step adaptivity proposed in Algorithm 2.2 can improve execution time, and that our algorithm performs faster than the iPiano algorithm in certain instances.

There are still open questions, two of which carry special importance. First, it should be possible to improve upon the time step adaptivity of Algorithm 2.2 which, while effective in some instances, is rudimentary. It may be possible to employ a stochastically ordered version as in [19] instead. Second, one should investigate the convergence properties of Algorithm 2.1 when applied to nondifferentiable problems since it is still applicable then. One may also generalize Algorithm 2.1 to a manifold setting using the tools developed in [7]. Finally, one may wish to apply the discrete gradient approach to other nonconvex optimization problems.

Acknowledgments. The authors wish to thank Antonin Chambolle, Elena Celledoni, Brynjulf Owren, and Reinout Quispel for their helpful discussions during this work. We also wish to thank the anonymous referees for their thorough advice.

REFERENCES

[1] E. Bae, X.-C. Tai, and W. Zhu, *Augmented Lagrangian method for an Euler's elastica based segmentation model that promotes convex contours*, Inverse Probl. Imag., 11 (2017), pp. 1–23.

[2] A. Beck and L. Tetruashvili, *On the convergence of block coordinate descent type methods*, SIAM J. Optim., 23 (2013), pp. 2037–2060.

[3] D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific Belmont, MA, 1999.

[4] J. Bolte, S. Sabach, and M. Teboulle, *Proximal alternating linearized minimization for nonconvex and nonsmooth problems*, Math. Program., 146 (2014), pp. 459–494.

[5] K. Bredies, T. Pock, and B. Wirth, *A convex, lower semicontinuous approximation of Euler's elastica energy*, SIAM J. Math. Anal., 47 (2015), pp. 566–613.

[6] R. P. Brent, *An algorithm with guaranteed convergence for finding a zero of a function*, Comput. J., 14 (1971), pp. 422–425.

[7] E. Celledoni and B. Owren, *Preserving first integrals with symmetric Lie group methods*, Discrete Contin. Dyn. Syst., 34 (2014), pp. 977–990.

[8] A. Chambolle and T. Pock, *A first-order primal-dual algorithm for convex problems with applications to imaging*, J. Math. Imaging Vis., 40 (2011), pp. 120–145.

[9] A. Chambolle and T. Pock, *Total Roto-translational Variation*, arXiv:1709.09953, 2017.

[10] M. J. Ehrhardt, E. S. Riis, T. Ringholm, and C.-B. Schönlieb, *A Geometric Integration Approach to Smooth Optimisation: Foundations of the Discrete Gradient Method*, arXiv:1805.06444, 2018.

[11] O. Gonzalez, *Time integration and discrete Hamiltonian systems*, J. Nonlinear Sci., 6 (1996), pp. 449–467.

[12] V. Grimm, R. I. McLachlan, D. I. McLaren, G. Quispel, and C. Schönlieb, *Discrete gradient methods for solving variational image regularisation models*, J. Phys. A, 50 (2017), p. 295201.

[13] E. Hairer and C. Lubich, *Energy-diminishing integration of gradient systems*, IMA J. Numer. Anal., 34 (2013), pp. 452–461.

[14] A. Harten, P. D. Lax, and B. van Leer, *On upstream differencing and Godunov-type schemes for hyperbolic conservation laws*, SIAM Rev., 25 (1983), pp. 35–61.

[15] T. Itoh and K. Abe, *Hamiltonian-conserving discrete canonical equations based on variational difference quotients*, J. Comput. Phys., 76 (1988), pp. 85–102.

[16] H. Karimi, J. Nutini, and M. Schmidt, *Linear convergence of gradient and proximal-gradient methods under the Polyak–Łojasiewicz condition*, in Proceedings of ECML-PKDD, Springer, New York, 2016, pp. 795–811.

[17] R. I. McLachlan, G. R. W. Quispel, and N. Robidoux, *Geometric integration using discrete gradients*, Philos. Trans. A, 357 (1999), pp. 1021–1045.

[18] Y. Nesterov, *A method of solving a convex programming problem with convergence rate* $\mathcal{O}(1/k^2)$, Soviet Math. Dokl., 27 (1983), pp. 372–376.

[19] Y. Nesterov, *Efficiency of coordinate descent methods on huge-scale optimization problems*, SIAM J. Optim., 22 (2012), pp. 341–362.

[20] Y. Nesterov, *Gradient methods for minimizing composite functions*, Math. Program., 140 (2013), pp. 125–161.

[21] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*, Stud. Appl. Numer. Math. 13, SIAM, Philadelphia, 1994.

[22] M. Nitzberg, D. Mumford, and T. Shiota, *Filtering, Segmentation and Depth*, Lecture Notes in Comput. Sci., Springer, New York, 1993.

[23] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer Ser. Oper. Res. Financ. Eng., Springer, New York, 2006.

[24] P. Ochs, Y. Chen, T. Brox, and T. Pock, *iPiano: Inertial proximal algorithm for nonconvex optimization*, SIAM J. Imaging Sci., 7 (2014), pp. 1388–1419.

[25] B. T. Polyak, *Some methods of speeding up the convergence of iteration methods*, USSR Comp. Math. Math+, 4 (1964), pp. 1–17.

[26] W. H. Press, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed., Cambridge university Press, Cambridge, UK, 2007.

[27] E. S. Riis, M. J. Ehrhardt, G. Quispel, and C.-B. Schönlieb, *A Geometric Integration Approach to Nonsmooth, Nonconvex Optimisation*, arXiv:1807.07554, 2018.

[28] L. I. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Phys. D, 60 (1992), pp. 259–268.

[29] D. Scieur, V. Roulet, F. Bach, and A. d'Aspremont, *Integration methods and optimization algorithms*, in Advances in Neural information Processing Systems 30, Curran Associates, New York, 2017, pp. 1109–1118.

[30] J. Shen, S. Kang, and T. Chan, *Euler's elastica and curvature-based inpainting*, SIAM J. Appl. Math., 63 (2003), pp. 564–592.

[31] N. Z. Shor, *Minimization Methods for Non-differentiable Functions*, Springer, New York, 1985.

[32] X.-C. Tai, J. Hahn, and G. J. Chung, *A fast algorithm for Euler's elastica model using augmented Lagrangian method*, SIAM J. Imaging Sci., 4 (2011), pp. 303–344.

[33] P. Wolfe, *Convergence conditions for ascent methods*, SIAM Rev., 11 (1969), pp. 226–235.

[34] S. J. Wright, *Coordinate descent algorithms*, Math. Program., 151 (2015), pp. 3–34.

[35] D. Xie and L. Adams, *New parallel SOR method by domain partitioning*, SIAM J. Sci. Comput., 20 (1999), pp. 2261–2281.

[36] J. Zhang and K. Chen, *A new augmented Lagrangian primal dual algorithm for elastica regularization*, J. Algorithms Comput. Technol., 10 (2016), pp. 325–338.

[37] J. Zhang, R. Chen, C. Deng, and S. Wang, *Fast linearized augmented Lagrangian method for Euler's elastica model*, Numer. Math. Theory Methods Appl., 10 (2017), pp. 98–115.

[38] W. Zhu, X.-C. Tai, and T. Chan, *Augmented Lagrangian method for a mean curvature based image denoising model*, Inverse Probl. Imag., 7 (2013), pp. 1409–1432.