A **<u>Flex</u>**ible Primal-Dual Tool**<u>Box</u>**

Technical Report

Hendrik Dirks*

February 25, 2016

Abstract

FlexBox is a flexible MATLAB toolbox for finite dimensional convex variational problems in image processing and beyond. Such problems often consist of non-differentiable parts and involve linear operators. The toolbox uses a primal-dual scheme to avoid (computationally) inefficient operator inversion and to get reliable error estimates. From the user-side, FlexBox expects the primal formulation of the problem, automatically decouples operators and dualizes the problem. For large-scale problems, FlexBox also comes with a C++-module which can be used stand-alone or together with MATLAB via mex-interfaces. Besides various pre-implemented data-fidelity and regularization-terms, FlexBox is able to handle arbitrary operators while being easily extendable, due to its object-oriented design.

1 Introduction

Many variational problems in image processing can be written in the form

$$\arg\min_x G(x) + F(Ax), \tag{1}$$

where A denotes some linear operator and both G and F are proper, convex and lower-semicontinuous functions (see 4.2). Problem (1) refers to the so-called *primal* formulation of the minimization problem and x is known as the primal variable. It can be shown (see [11]) that minimizing (1) is equivalent to minimizing the *primal-dual* or *saddle-point* formulation

$$\arg\min_x \arg\max_y G(x) + \langle y, Ax\rangle - F^*(y). \tag{2}$$

Here, F^* refers to the convex conjugate (see 4.2) of F and y will be denoted as the dual variable. It the recent years, algorithms like ADMM [8, 14] or

*Institute for Computational and Applied Mathematics and Cells in Motion Cluster of Excellence, University of Münster, Orléans-Ring 10, 48149 Münster, Germany, Email: hendrik.dirks@wwu.de

primal-dual [10, 3, 17, 5] for efficiently solving these saddle-point problems have become very popular. **FlexBox** makes use the latter, which can be sketched up as follows:
For $\tau, \sigma > 0$ and a pair $(\hat{x}^0, y^0) \in \mathcal{X} \times \mathcal{Y}$ we iteratively solve:

$$y^{k+1} = prox_{\sigma F^*}(y^k + \sigma A \hat{x}^k) \tag{3}$$
$$x^{k+1} = prox_{\tau G}(x^k - \tau A^T y^{k+1}) \tag{4}$$
$$\hat{x}^{k+1} = 2x^{k+1} - x^k \tag{5}$$

Here, $prox_{\tau G}$ denotes the *proximal* or *resolvent* operator

$$prox_{\tau G}(y) = (I + \tau \partial G)^{-1}(y) := \arg\min_v \left\{ \frac{\|v - y\|_2^2}{2} + \tau G(v) \right\},$$

which can be interpreted as a compromise between minimizing G and being close to the input argument y. The efficiency of primal-dual algorithms relies on the fact that the *prox-problems* are computationally efficient to solve.

1.1 Contribution

Since primal-dual algorithms have been extensively applied to all classes of convex optimization problems, we found that people are spending a lot of effort on calculating convex conjugates or solutions for prox-problems again and again for similar problems. Let us consider, for example, the isotropic total variation $\|\nabla u\|_{1,2}$, where the convex conjugate is an indicator function of the L^∞-ball and the solution of the prox-problem is a point-wise projection onto L^2-balls. These results hold not only for $A = \nabla$, but for arbitrary operators. **FlexBox** makes use of this generalization and simply works on the level of terms in the primal problem. After adding a certain term, **FlexBox** automatically decouples operators, creates dual variables and calculates step-sizes (τ, σ). **FlexBox** already contains a variety of data-fidelity (e.g. L^1, L^2, Kullback-Leibler) and regularization terms (e.g. L^2, TV, Laplace, curl), but is also compatible with user-defined operators. Moreover, the class-based structure allows easy extension and creation of custom terms. A full list of available terms can be found in Table 4.2.
Core components of **FlexBox** are written in MATLAB, but there exists an optional C++-module to improve compatibility and runtime. This module can be compiled and will afterwards be used automatically via a MEX-interface. The C++-module can also be used without MATLAB (but with e.g. OpenCV [1, 2]), but does not have the full functionality and variety of terms included.

2 Architecture and Features

Basic Idea: To work with **FlexBox**, the user has to derive the primal formulation of the variational problem first. Afterwards, each primal variable including its dimensions is added to **FlexBox**. Then, the primal problem is put into the toolbox term by term, using the implemented functional terms from Table 4.2. Once the computation is finished, the result, stored in the primal variables, can be requested and used.

Design: **FlexBox** is designed as one core class which holds a list of functional terms. Those terms are internally specified either as type *primal* or type *dual*. The main object holds the data of primal and dual variables x_i and y_i and all parameters. Terms always correspond to at least one primal variable, whereas dual terms also correspond at least one dual variable and contain the involved operators. **FlexBox** automatically creates necessary dual variables once a dual term is added.
In the primal-dual algorithm (5), applications of the operator can be decoupled defining

$$\tilde{y} := y^k + \sigma A\hat{x}^k, \text{ and } \tilde{x} := x^k - \tau A^T y^{k+1}.$$

Since dual terms bind the operators, calculations of $\tilde{y}$ and $\tilde{x}$ are done inside the corresponding dual terms, accessing the variables held by the core class. Finally, primal and dual terms specify prox-methods to solve the arising prox-problems while again accessing the variables held by the **FlexBox** core class.

Parameters: To ensure convergence, the parameters τ and σ have to fulfill $\tau\sigma\|A\| < 1$. A static choice of these parameters might lead to slow convergence speed, because it is dominated by the worst possible values along all primal and dual variables. A fully automatic strategy for finding *optimal* parameters can be extrapolated from [9]. Summing up the absolute values of row elements (for σ_i) resp. column elements (for τ_i) leads to custom parameters for each term in the functional. This strategy is inherited by **FlexBox**.

Stopping Criterion: As a powerful stopping criterion **FlexBox** uses the primal-dual residual, proposed by Goldstein, Esser and Baraniuk [6] which can be calculated after the (k+1)-th iteration as

$$p^k := \left| \frac{x^k - x^{k+1}}{\tau} - A^T(y^k - y^{k+1}) \right|, \quad d^k := \left| \frac{y^k - y^{k+1}}{\sigma} - A(x^k - x^{k+1}) \right|,$$

with $|\cdot|$ being the sum of absolute values. We denote p^k as the primal and d^k as the dual residual. The total residual is then given by the sum of primal- and dual residual which is afterwards scaled with the size of the problem and number of variables. Since evaluating the residual is computationally expensive, it is regularly computed after a fixed number of iterations (default 100).
Besides the primal-dual residual, **FlexBox** automatically stops after a static number of iterations (default 10000) and can be continued afterwards.

3 Examples

3.1 Rudin-Osher-Fatemi

The Rudin-Osher-Fatemi model [12] has very popular applications in image denoising. The primal formulation reads

$$\arg\min_{u} \frac{1}{2}\|u-f\|_2^2 + \alpha\|\nabla u\|_{1,2}, \tag{6}$$

where the first part fits the unknown u to the given input image f and the second part refers to the isotropic total variation, which penalizes the total number of jumps in the solution. Minimizing this problem with **FlexBox** can be done with the following lines of code

```
1  %Begin: Code example
2  main = flexbox;
3
4  numberU = main.addPrimalVar(size(f));
5
6  main.addTerm(L2dataTerm(1,imageNoisy),numberU);
7  main.addTerm(L1gradientIso(0.08,size(f)),numberU);
8
9  main.runAlgorithm;
10
11 result = main.getPrimal(numberU);
12 %End: Code example
```

To keep this example short, we omited parts of the code where the image f is read. Let us begin in line 2, which initializes a **FlexBox** object and saves it to the variable $main$. Line 4 then adds the primal objective variable u which has the same size as the input image f. The toolbox returns the internal number of this primal variable, which is saved in $numberU$.
In line 6 and 7, the L^2-data-term with weight $\alpha = 1$ (the weight is divided by 2 internally) and corresponding image f is added, moreover the isotropic TV-term with weight 0.08 is pushed into the framework. Note that the function $addTerm$ always requires a functional part and the internal number of the corresponding primal variable.
The function call in line 9 finally starts the calculation and once this is finished we transfer the solution into the variable $result$ in line 11.

3.2 Optical Flow

Estimating the motion between two consecutive images f_1 and f_2 based on the displacement of intensities in both images is called optical-flow estimation. The unknown velocity field $\boldsymbol{v}$ is usually connected to the image by the brightness-constancy-assumption $f_2(x+\boldsymbol{v}) - f_1(x) = 0$. This formulation is non-linear in terms of $\boldsymbol{v}$ and therefore linearized (see e.g. [16, 4]). A corresponding variational problem incorporating total variation regularization can be written as

$$\underset{\boldsymbol{v}=(v_1,v_2)}{\arg\min}\ \frac{1}{2}\|f_2 - f_1 + \nabla f_2 \cdot \boldsymbol{v}\|_1 + \alpha_1\|\nabla v_1\|_{1,2} + \alpha_2\|\nabla v_2\|_{1,2}, \tag{7}$$

Solving this problem with **FlexBox** can be done in a similar manner as for the ROF example:

```
1  %Begin: Code example
2  main = flexbox;
3
4  numberV1 = main.addPrimalVar(size(f1));
5  numberV2 = main.addPrimalVar(size(f2));
6
7  %add optical flow data term
8  main.addTerm(L1opticalFlowTerm(1,f1,f2),[numberV1,numberV2]);
9
10 %add regularizers - one for each component
11 main.addTerm(L1gradientIso(0.05,size(f1)),numberV1);
12 main.addTerm(L1gradientIso(0.05,size(f1)),numberV2);
13
14 main.runAlgorithm;
15
16 resultV1 = main.getPrimal(numberV1);
17 resultV2 = main.getPrimal(numberV2);
18 %End: Code example
```

In lines 4 and 5 primal variables for both components of the velocity fields are added. Afterwards, in lines 8, 11 and 12 the data term and regularizers for both components are inserted. Please note that the optical flow term now refers to two primal variables written as the vector $[numberV1, numberV2]$. Afterwards, the algorithm is started and both results are retrieved.

3.3 Segmentation

Dividing an image into different regions is called *segmentation*. Assuming that the image f consists of k different regions, each of them having a mean intensity c_i, the segmentation problem including total variation regularization can be written as

$$\underset{u=u_1,\dots,u_k}{\arg\min} \sum_{i=1}^{k} u_i \frac{1}{2}\|f - c_i\|_2^2 + \alpha\|\nabla u_i\|_{1,2}, \tag{8}$$

$$s.t. \quad u_i \geq 0, \quad \sum_{i=1}^{k} u_i = 1 \tag{9}$$

where u is a labeling vector (see [7, 15]). This labeling formulation is a convex relaxation of the integer assignment $u_i \in \{0, 1\}$. The MATLAB implementation of this problem is again rather short:

```
1  %Begin: Code example
2  numberOfLabels = 3;
3  dims = size(image);
4  labels = rand(numberOfLabels,1);
5
6  main = flexbox;
7
8  for i=1:numberOfLabels
9      main.addPrimalVar(size(image));
10 end
11
12 %init data term
13 main.addTerm(labelingTerm(1,image,labels),1:numberOfLabels);
14
15 for i=1:numberOfLabels
16     main.addTerm(L1gradientIso(0.5,size(image)),i);
17 end
18
19 main.runAlgorithm;
20
21 for i=1:numberOfLabels
22     labelMatrix(:,:,i) = main.getPrimal(i);
23 end
24 %End: Code example
```

We begin by choosing a fixed number of regions and choose the *mean* intensity in each region as random values in line 4. Afterwards, the main object is initialized and a primal variable for each u_i is created in lines 8-10. In line 13, we create the labeling term using the previously defined labels and creating a connection for primal variables *1:numberOfLabels*. The loop in line 13-15 creates a total variation regularizer for each of the primal variables. The problem is solved in line 19 and we save each labeling function as a layer in a 3d matrix.

4 Features

4.1 General

- **FlexBox** can be stopped at any time and afterwards continued from the current state. Due to this feature, we are able to change parameters after the problem has converged and use the current state as an initial guess for the next run.

- The toolbox supports arbitrary user-defined operators consisting of blocks A_i (see *General operator regularization* in Table 4.2) that can be submitted by defining A as a cell-array of blocks where each *numPrimals* elements correspond to one row in the overall operator.

4.2 Available Terms

List of implemented terms

Term	Classname	Parameters	
Data-fidelity			
$\alpha\|u-f\|_1$	L1dataTerm(alpha,f)	f: α:	input data weight
$\alpha\|Au-f\|_1$	L1dataTermOperator(alpha,A,f)	f: α: A:	input data weight operator
$\frac{\alpha}{2}\|u-f\|_2^2$	L2dataTerm(alpha,f)	f: α:	input data weight
$\frac{\alpha}{2}\|Au-f\|_2^2$	L2dataTermOperator(alpha,A,f)	f: α: A:	input data weight operator
$\int Au - f + f\log\frac{f}{Au}$ s.t. $u \geq 0$	KLdataTerm(alpha,A,f)	f: α: A:	input data weight operator
$\alpha\|\nabla f_2 \cdot \boldsymbol{v} + f_2 - f_1\|_1$	L1opticalFlowTerm(alpha,f_1,f_2)	f_1,f_2: α:	images weight
$\frac{\alpha}{2}\|\nabla f_2 \cdot \boldsymbol{v} + f_2 - f_1\|_2^2$	L2opticalFlowTerm(alpha,f_1,f_2)	f_1,f_2: α:	images weight
$\sum_{i=1}^{n}\langle u_i, f_i\rangle$ s.t. $u_i \geq 0, \sum u_i = 1$ s.t. $f_i = (f-l_i)^2$	labelingTerm(alpha,f,l)	f: α: l: dims:	image weight label vector dimensions of u
Gradient regularization			
$\alpha\|\nabla u\|_{1,1}$	L1gradientAniso(alpha,dims)	α: dims:	weight dimensions of u
$\alpha\|\nabla u\|_{1,2}$	L1gradientIso(alpha,dims)	α: dims:	weight dimensions of u
$\frac{\alpha}{2}\|\nabla u\|_2^2$	L2gradient(alpha,dims)	α: dims:	weight dimensions of u
$\alpha\|\nabla u\|_{H_\epsilon}$	huberGradient(alpha,dims,epsi)	α: dims: epsi:	weight dimensions of u ϵ for Huber-norm
$\alpha\|\nabla u\|_F$	frobeniusGradient(alpha,dims)	α: dims:	weight dimensions of u
$\alpha\|\nabla(u-w)\|_{1,1}$	L1gradientDiffAniso(alpha,dims)	α: dims:	weight dimensions of u
$\alpha\|\nabla(u-w)\|_{1,2}$	L1gradientDiffIso(alpha,dims)	α: dims:	weight dimensions of u
$\alpha\|\nabla u - w\|_{1,1}$	L1secondOrderGradientAniso(alpha,dims)	α: dims:	weight dimensions of u
$\alpha\|\nabla u - w\|_{1,2}$	L1secondOrderGradientIso(alpha,dims)	α: dims:	weight dimensions of u
General operator regularization			

$\alpha\|Au\|_{1,1}$	L1operatorAniso(alpha,numPrimals,A)	α: weight numPrimals: # corresp. primals A: operator(s)
$\alpha\|Au\|_{1,2}$	L1operatorIso(alpha,numPrimals,A)	α: weight numPrimals: # corresp. primals A: operator(s)
$\frac{\alpha}{2}\|Au\|_2^2$	L2operator(alpha,numPrimals,A)	α: weight numPrimals: # corresp. primals A: operator(s)
$\alpha\|Au\|_F$	frobeniusOperator(alpha,numPrimals,A)	α: weight numPrimals: # corresp. primals A: operator(s)
$\alpha\langle b, Au\rangle$	innerProductOperator(alpha,A,b)	α: weight A: operator b: vector b
	Vector-field regularization	
$\alpha\|curl(\boldsymbol{v})\|_1$	L1curl(alpha,dims)	α: weight dims: dimensions of e.g. v_1
$\frac{\alpha}{2}\|curl(\boldsymbol{v})\|_2^2$	L2curl(alpha,dims)	α: weight dims: dimensions of e.g. v_1
$\alpha\|\nabla\cdot\boldsymbol{v}\|_1$	L1divergence(alpha,dims)	α: weight dims: dimensions of e.g. v_1
$\frac{\alpha}{2}\|\nabla\cdot\boldsymbol{v}\|_2^2$	L2divergence(alpha,dims)	α: weight dims: dimensions of e.g. v_1
	Other regularization	
$\alpha\|u\|_{1,1}$	L1identity(alpha,dims)	α: weight dims: dimensions of u
$\frac{\alpha}{2}\|u\|_2^2$	L2identity(alpha,dims)	α: weight dims: dimensions of u
$\alpha\langle b, \nabla u\rangle$	innerProductGradient(alpha,dims,b)	α: weight dims: dimensions of u b: vector b
$\alpha\langle b, \nabla(u-w)\rangle$	innerProductGradientDiff(alpha,dims,b)	α: weight dims: dimensions of u b: vector b

Table 1: List of terms currently available in **FlexBox**.

A General Considerations

- Throughout this report we consider finite dimensional optimization problems with primal variables $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- The sets $\mathcal{X} \subset \mathbb{R}^N$ and $\mathcal{Y} \subset \mathbb{R}^M$ are assumed to be convex.

- We do not explicitly distinguish between a variable x_c on a regular cartesian grid $N_1 \times \ldots \times N_d$ (s.t. $N_1 \cdot \ldots \cdot N_d = N$) and its vectorized equivalent $x \in \mathbb{R}^N$, which is gained by concatenating x along the first dimension.

- The scalar product of two vectors on $\mathcal{X}$ is defined as

$$\langle u, v \rangle = \sum_i u_i v_i, \quad u, v \in \mathcal{X}.$$

- **FlexBox** expects the matrix representation of the linear operator A, hence $A \in \mathbb{R}^{M \times N}$ and $A^* = A^T$. Note that the evaluation of a linear operator can always be written as a matrix-vector multiplication by applying the Kronecker product (see e.g. [13]).

- The function F^* refers to the convex conjugate or Legendre-Fenchel transformation of F and is defined as

$$F^*(y^*) := \sup_{y \in \mathcal{Y}} \langle y^*, y \rangle - F(y).$$

Acknowledgements

- Thanks to Eva-Maria Brinkmann, Janic Föcke, Julian Rasch and Carolin Roßmanith for testing **FlexBox** and discussing useful features.

- **FlexBox** makes use of the primal-dual algorithm working on concrete functional terms. A MATLAB toolbox consisting of a more general class of solvers for optimization problems is OOMFIP, developed by Martin Benning.

References

[1] Gary Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[2] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[3] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[4] Hendrik Dirks. *Variational Methods for Joint Motion Estimation and Image Reconstruction.* PhD thesis, WWU Münster, 2015.

[5] Ernie Esser, Xiaoqun Zhang, and Tony F Chan. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046, 2010.

[6] Tom Goldstein, Min Li, Xiaoming Yuan, Ernie Esser, and Richard Baraniuk. Adaptive primal-dual hybrid gradient methods for saddle-point problems. *arXiv preprint arXiv:1305.0546*, 2013.

[7] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on pure and applied mathematics*, 42(5):577–685, 1989.

[8] Neal Parikh and Stephen P Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014.

[9] Thomas Pock and Antonin Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1762–1769. IEEE, 2011.

[10] Thomas Pock, Daniel Cremers, Horst Bischof, and Antonin Chambolle. An algorithm for minimizing the mumford-shah functional. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1133–1140. IEEE, 2009.

[11] Ralph Tyrell Rockafellar. *Convex analysis*. Princeton university press, 2015.

[12] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[13] Willi-Hans Steeb. *Kronecker product of matrices and applications*. BI-Wissenschaftsvlg, 1991.

[14] Bo Wahlberg, Stephen Boyd, Mariette Annergren, and Yang Wang. An admm algorithm for a class of total variation regularized estimation problems. *arXiv preprint arXiv:1203.1828*, 2012.

[15] Christopher Zach, David Gallup, Jan-Michael Frahm, and Marc Niethammer. Fast global labeling for real-time stereo using multiple plane sweeps. In *VMV*, pages 243–252, 2008.

[16] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Pattern Recognition*, pages 214–223. Springer, 2007.

[17] Xiaoqun Zhang, Martin Burger, and Stanley Osher. A unified primal-dual algorithm framework based on bregman iteration. *Journal of Scientific Computing*, 46(1):20–46, 2011.