A **Flex**ible Primal-Dual Tool**Box**

Technical Report

Hendrik Dirks

February 4, 2016

Abstract

Abstract

1 Introduction

Many variational problems can be written in the form

$$\arg\min_u G(u) + F(Ku), \tag{1}$$

which is generally denoted as the *primal* formulation of the minimization problem. It can be shown that minimizing (1) is equivalent to minimizing the *primal-dual* or *saddle-point* formulation

requirements

$$\arg\min_u \arg\max_p G(u) + \langle y, Ku\rangle - F^*(y). \tag{2}$$

Here F^* refers to the convex conjugate of F. It the recent years, algorithms like ADMM or primal-dual for efficiently solving these saddle-point problems have become very popular. **FlexBox** makes use the latter, which can be sketched up as follows:

ref

For $\tau, \sigma > 0$, a pair $(v^0, y^0) \in \mathcal{X} \times \mathcal{Y}$ and initial value $\bar{v}^0 = 0$ we obtain the following iterative scheme:

$$y^{k+1} = prox_{\sigma F^*}(y^k + \sigma K \hat{x}^k) \tag{3}$$

$$x^{k+1} = prox_{\tau G}(x^k - \tau K^* y^{k+1}) \tag{4}$$

$$\hat{x}^{k+1} = 2x^{k+1} - x^k \tag{5}$$

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 easy to compute.

1.1 Contibution

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 $K = \nabla$, 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 (e.g. τ, σ). **FlexBox** already contains a variety of data-fidelity and regularization terms, but is also with to user-defined operators. Moreover, the class-based structure allows easy extension and creation custom terms.

REF zu Liste an Termen

FlexBox is mainly written in MALTAB, but has an optional C++module to improve runtime. This module can be compiled and is afterwards automatically used via a mex-interface. The C++module can also be used without Matlab, but does not yet have the full variety of terms included.

2 Architecture and Features of FlexBox

Design: **FlexBox** is designed as one main class which holds a list of functional terms that are either *primal* or *dual*. Moreover, the main object holds the data of all primal and dual variables x_i and y_i. Primal and dual terms always correspond to at least primal variable, whereas dual terms also correspond at least one dual variable. **FlexBox** automatically creates dual variables once a dual term is added. In the primal-dual algorithm (5), applications of the operator can be decoupled by defining $\tilde{y} := y^k + \sigma K \hat{x}^k$ and $\tilde{x} := x^k - \tau K^* y^{k+1}$

Both, primal and dual terms, contain a corresponding prox-method

3 Examples

3.1 Rudin-Osher-Fatemi

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

REF

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

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 jumps in the solution. Minimizing this problem with FlexBox can be done with the following lines of code

```
%Begin: Code example
main = flexbox;

numberU = main.addPrimalVar(size(f));

main.addTerm(L2dataTerm(0.5,f),numberU);
main.addTerm(L1gradientIso(0.08,size(f)),numberU);

main.runAlgorithm;

result = main.getPrimal(numberU);
%End: Code example
```

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 0.5 and corresponding image f is added, moreover the isotropic TV-term with weight 0.08 is pushed into the framework. Please mention 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. A corresponding variational problem incorporating total variation regularization can be written as

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

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

```
%Begin: Code example
main = flexbox;

numberV1 = main.addPrimalVar(size(f1));
numberV2 = main.addPrimalVar(size(f2));

%add optical flow data term
main.addTerm(L1opticalFlowTerm(1,f1,f2),[numberV1,numberV2]);

%add regularizers - one for each component
main.addTerm(L1gradientIso(0.05,size(f1)),numberV1);
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 Labeling

4 How-To

5 Arbitrary Operators

Classes: *L1operatorIso, L1operatorAniso, L2operator, frobeniusOperator*
Adding an arbitrary operator in some norm is simple. Let us assume we some term $\alpha\|Ku\|_{1,2}$ with

$$K = \begin{pmatrix} K_1 & K_2 \\ K_3 & K_4 \end{pmatrix}, u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix},$$

where the norm $\|\cdot\|_{1,2}$ refers to the isotropic L1-norm. We can insert this term into **FlexBox** by calling

```
1 %Begin: Code example
2 main.addTerm(L1operatorIso(alpha,2,{K_1,K_2,K_3,K_4}),[numberU_1,numberU_2]);
3 %End: Code example
```

The operator K has to be specified row-wise in a cell-array. The second argument 2 tells the toolbox that every two elements in the cell-array correspond to one row. Please note that empty blocks in K have to be specified as empty sparse matrices.

A Mathematical Setup

B Prox Calculation

B.1 Frobenius-Norm

Minimization problem

$$\min_u \alpha\|Ku\|_F \tag{8}$$

Decuopling the operator K reads

$$F(\tilde{u}) = \alpha\|\tilde{u}\|_F \tag{9}$$

Calculating convex conjugate (Frobenius-norm is self-adjoint)

$$F^*(y) = \delta_{\{\|y/\alpha\|_{F^*}\leq 1\}} = \delta_{\{\|y/\alpha\|_F\leq 1\}} \tag{10}$$

Prox reads:

$$prox_{\sigma F^*}(\tilde{y}) = \min_y \|y-\tilde{y}\|_2^2 + \delta_{\{\|y/\alpha\|_F\leq 1\}} \tag{11}$$

Some calculation yields the compact solution

$$prox_{\sigma F^*}(\tilde{y}) = \frac{\tilde{y}}{\max\{1, \|\tilde{y}/\alpha\|_F\}} \tag{12}$$

B.2 Kullback-Divergence

Minimization problem

Calculate with factor alpha

$$\min_u Ku - f + f\log\frac{f}{Ku} + \delta_{\{\cdot\geq 0\}}(Ku) \tag{13}$$

Decuopling the operator reads

$$F(\tilde{u}) = \tilde{u} - f + f\log\frac{f}{\tilde{u}} + \delta_{\{\cdot\geq 0\}}(\tilde{u}) \tag{14}$$

Calculating convex conjugate

$$F^*(y) = -f\log(\mathbf{1}-y) + \delta_{\{y\geq 0\}}(\mathbf{1}-y) \tag{15}$$

Prox reads:

$$prox_{\sigma F^*}(\tilde{y}) = \min_y \frac{1}{2}\|y-\tilde{y}\|_2^2 - \sigma f\log(\mathbf{1}-y) + \delta_{\{y\geq 0\}}(\mathbf{1}-y) \tag{16}$$

Some calculation yields the compact solution

$$prox_{\sigma F^*}(\tilde{y}) = \frac{1}{2}\left(\mathbf{1}+\tilde{y} - \sqrt{(\tilde{y}-\mathbf{1})^2 + 4\sigma f}\right) \tag{17}$$

B.3 Inner Products

$$\min_u \alpha\langle Ku, b\rangle \tag{18}$$

Decoupling the operator reads

$$F(\tilde{u}) = \langle\tilde{u}, \alpha b\rangle \tag{19}$$

Calculating convex conjugate

$$F^*(y) = \sup_{\tilde{u}} \langle \tilde{u}, y \rangle - \langle \tilde{u}, \alpha b \rangle = \sup_{\tilde{u}} \langle \tilde{u}, y - \alpha b \rangle = \begin{cases} 0 & \text{for } y = \alpha b \\ \infty & \text{else} \end{cases} \tag{20}$$

Prox reads:

$$prox_{\sigma F^*}(\tilde{y}) = \min_y \frac{1}{2} \|y - \tilde{y}\|_2^2 + \sigma \delta_{\{y = \alpha b\}} \tag{21}$$

Compact solution

$$prox_{\sigma F^*}(\tilde{y}) = \alpha b \tag{22}$$

Todo list