Guide to the MATLAB code for total variation denoising with the DAM algorithm

Amir Beck, Ariel Shem-Tov, Luba Tetruashvili and Yakov Vaisbourd

The Dual Block Coordinate - Total Variation (DBC-TV) package available with this documentation note is a MATLAB implementation of the Dual Alternating Minimization (DAM) method presented in the paper:

> [BSTV16] A. Beck, L. Tetruashvili, Y. Vaisbourd, A. Shemtov, "Rate of convergence analysis of dual-based variables decomposition methods for strongly convex problems", Operations Research Letters, Volume 44, Issue 1, January 2016, Pages 61−66,

for solving the Total Variation (TV) denoising problem:

$$\min_{\mathbf{x}\in\mathbb{R}^{m\times n}} \frac{1}{2}\|\mathbf{x}-\mathbf{b}\|_F^2 + \theta\cdot \mathrm{TV}(\mathbf{x}), \tag{TV}$$

where $\mathbf{b}$ is the observed noisy image, $\mathbf{x}$ is the "true" image to be recovered, $\theta > 0$ is a regularization parameter and $\mathrm{TV}(\cdot)$ stands for the discrete TV. We consider the isotropic (TV_I) and the anisotropic (TV_{l_1}) TV.
Note that for the TV denoising problem, the sequences generated by both DAM and DBPG algorithms presented in [BSTV16] are identical. Hence, this package includes only the implementation of the DAM method.

The package contains the following m-files:

1. `dbc_tv.m` - an implementation of the DAM algorithm for solving the TV denoising problem.
2. `decompose.m` - a function that decomposes a given 1D signal or a 2D image into several sets of separable blocks.
3. `sp_newton.m` - an implementation of the root finding Newton method for solving the sub-problems that emerge in the TV_I setting.
4. `example.m` - a short example that demonstrates the usage of the package.

Note that the functions mentioned in (2) and (3) operate as sub-procedures called by the main function `dbc_tv`. Hence, in this note we will not provide further information regarding the usage of these functions and, in the case such details are of interest, we refer the reader to

the additional documentation that is available within the source code of the corresponding files. We now turn to describe the input and output arguments of the `dbc_tv` function.

Input:

Compulsory arguments:

- `d` - The observed 1D signal or 2D image contaminated by noise.

Optional arguments:

Each of the following arguments, if necessary, should be specified by a comma-separated `Name,Value` pair. "Name" is the argument name and "Value" is the corresponding value. Note that "Name" should be enclosed within single quotes (' ').

Argument Name	Description	Acceptable Values	Default Value
theta	The regularization parameter θ	positive real	0.1
TV	TV regularizer type; isotropic or anisotropic	'iso','l1'	'iso'
type	Index selection strategy	'cyclic','random'	'cyclic'
max_iter	Maximum number of iterations	positive integer	200
epsilon	Tolerance for relative error used in the stopping criteria	positive real	-
maxiter_subproblem	Maximum number of iterations used in the stopping criteria of the subproblems	positive integer	3
epsilon_subproblem	Tolerance for relative error used in the stopping criteria of the subproblems	positive real	-
output	Set to 1 in order to provide output or 0 in order to suppress it	1,0	0

Output:

- `x` - denoised signal/image which is the solution to (TV).
- `fun_val` - sequence of primal objective function values.

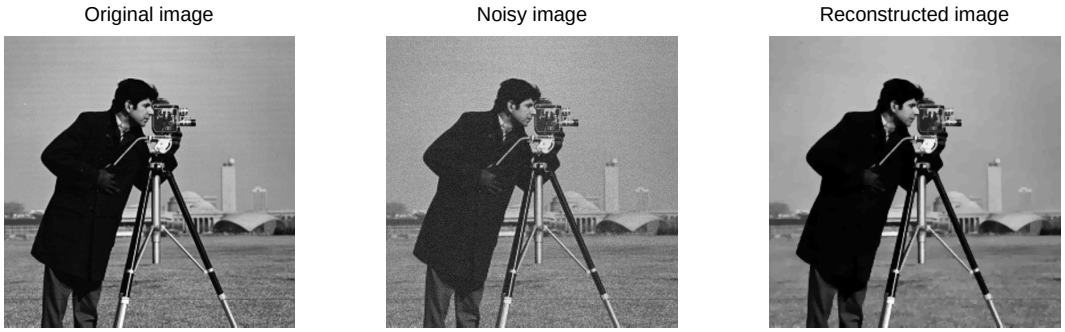
In the examples given below we consider the problem of denoising the 'cameraman' image contaminated with white Gaussian noise with standard deviation 0.05 generated by the following code:

```
randn('seed',316);
Img = double(imread('cameraman.tif'))/255;
noise_level = 0.05;
d = Img + noise_level*randn(size(Img));
```

- running `dbc_tv` with the default values and plotting the original, noisy and reconstructed images:

```
[x,f_val] = dbc_tv(d);
subplot(1,3,1), imagesc(Img), colormap gray, title('
   Original image'), ...
    axis square, axis off;
subplot(1,3,2), imagesc(d), colormap gray, title('Noisy
   image'), ...
    axis square, axis off;
subplot(1,3,3), imagesc(x), colormap gray, title('
   Reconstructed image'),...
    axis square, axis off;
```

The resulting images are given by



- running 50 iterations of DAM-r (the variation of DAM with a random index selection strategy) with $\theta = 0.05$ and the anisotropic TV regularizer:

```
[x,f_val] = dbc_tv(d,...
                   'type',    'random',...
                   'theta',    0.05,...
                   'TV',      'l1',...
                   'maxiter', 50);
```

- Running for 10 iterations and setting the output option to 1

```
[x,f_val] = dbc_tv(d,'maxiter',10,'output', 1);
```

will produce the following output:

```
Solving with DAM-c
#iteration  function-value  relative-difference
-----------------------------------------------
    0     2547.305011182858
    1     1032.518034496144        1514.786976686714
    2     862.777087498884         169.740946997260
    3     814.543254050854         48.233833448030
    4     791.710288091949         22.832965958905
    5     778.443524975280         13.266763116669
    6     769.616073697882         8.827451277398
    7     763.422987918561         6.193085779322
    8     758.869988116613         4.552999801947
    9     755.347066672417         3.522921444196
   10     752.560184634066         2.786882038351
```

The first column contains the iteration number, the second shows the corresponding primal function values and the last column contains the difference between the current and the previous iteration.

- The defaults of the root finding Newton algorithm can be also customized. For example, we can set the number of iterations to 1, hence providing an inexact solution to the sub-problems:

```
t_inexact = tic;
[x,f_val] = dbc_tv(d,'maxiter',20,'maxiter_subproblem',1);
tElap = toc(t_inexact);
disp([tElap,f_val(end)])
```

results with the output

```
    2.2061   740.9314
```

running the same method with the default number of Newton iterations:

```
t_exact = tic;
[x,f_val] = dbc_tv(d,'maxiter',20);
tElap = toc(t_exact);
disp([tElap,f_val(end)])
```

results with the output

```
    3.2101   740.4914
```

Hence, in this example, for a small sacrifice in the convergence rate we can substantially reduce the execution time of the algorithm.