Guide to the MATLAB code for total variation-based deblurring with FISTA

Amir Beck and Marc Teboulle

August 11, 2008

1 Overview

The MATLAB codes in this package are aimed at solving denoising problems of the form

$$\min_{\mathbf{X}}\{\|\mathbf{X}-\mathbf{B}\|^2+2\lambda\mathrm{TV}(\mathbf{X}):l\leq X_{ij}\leq u\}, \tag{1.1}$$

and deblurring problems of the form

$$\min_{\mathbf{X}}\{\|\mathcal{A}(\mathbf{X})-\mathbf{B}\|^2+2\lambda\mathrm{TV}(\mathbf{X}):l\leq X_{ij}\leq u\}, \tag{1.2}$$

where

- $\mathbf{B}\in\mathbb{R}^{m\times n}$ is the observed blurred and noisy image of size $m\times n$.
- $\mathcal{A}:\mathbb{R}^{m\times n}\to\mathbb{R}^{m\times n}$ is the linear operator corresponding to a spatially invariant PSF of the blur operation.
- $TV(\mathbf{X}):\mathbb{R}^{m\times n}\to\mathbb{R}$ is a total variation function. Here we consider two possibilities: the isotropic TV function:

$$\begin{array}{rcl}\mathbf{X}\in\mathbb{R}^{m\times n}, \quad \mathrm{TV}_I(\mathbf{X}) & = & \sum_{i=1}^{m-1}\sum_{j=1}^{n-1}\sqrt{(X_{i,j}-X_{i+1,j})^2+(X_{i,j}-X_{i,j+1})^2} \\ & & +\sum_{i=1}^{m-1}|X_{i,n}-X_{i+1,n}|+\sum_{j=1}^{n-1}|X_{m,j}-X_{m,j+1}|\end{array}$$

  and the l_1-based TV function:

$$\begin{array}{rcl}\mathbf{X}\in\mathbb{R}^{m\times n}, \quad \mathrm{TV}_{l_1}(\mathbf{X}) & = & \sum_{i=1}^{m-1}\sum_{j=1}^{n-1}\{|X_{i,j}-X_{i+1,j}|+|X_{i,j}-X_{i,j+1}|\} \\ & & +\sum_{i=1}^{m-1}|X_{i,n}-X_{i+1,n}|+\sum_{j=1}^{n-1}|X_{m,j}-X_{m,j+1}|.\end{array}$$

- λ - positive regularization parameter.

The functions are based on the paper

A. Beck and Marc Teboulle, "Fast Gradient-Based Algorithms for Constrained Total Variation Image Denoising and Deblurring Problems"

The package uses functions from the HNO package of Hansen, Nagy and O'leary available at

http://www2.imm.dtu.dk/~pch/HNO/

which is based on monograph [1]; thus the above package should be uploaded.
The package contains overall 7 functions.

- The denoising function

  `denoise_bound.m`

- The two deblurring functions

  `deblur_tv_fista.m`
  `deblur_tv_fista_sep.m`

- Four auxiliary functions

  `denoise_bound_init.`
  `Lforward.m`
  `Ltrans.m`
  `tlv.m`

2 Examples

2.1 Denoising

We begin by generating an artificial noisy image. For example, we can upload the camerman test image

```
>>X=double(imread('cameraman.pgm'));
>>X=X/255;
```

The second command is done for scaling reasons. The observed image is created by adding a white Gaussian noise with standard deviation 0.02.

```
>>randn('seed',314);
>>Bobs=X +2e-2*randn(size(X));
```

To see the original and noisy images we can write

Figure 1: original and noisy camraman

```
>> subplot(1,2,1)
>> imshow(X,[])
>> subplot(1,2,2)
>> imshow(Bobs,[])
```

The result can be seen in Figure 1.

To find the solution of (1.1) with $\lambda = 0.02$ and without constraints (that is, $l = -\infty, u = \infty$) we can invoke the function `deblur_bound.m`.

```
>>X_den=denoise_bound(Bobs,0.02,-Inf,Inf);
```

which gives the following output:

```
************************
*Solving with FGP/FISTA**
************************

 #iteration  function-value relative-difference
-----------------------------------------------
    1       18033.3983075949      1.0000000000
    2       18022.1438169810      0.0322767138
    3       18017.8747709622      0.0124521604
    4       18015.9814844837      0.0068134439
    5       18014.9973032615      0.0044135443
    6       18014.4157953842      0.0030907351
    7       18014.0419606571      0.0023226418
    8       18013.7874166548      0.0018098027
    9       18013.6030199019      0.0014670563
   10       18013.4638592325      0.0012121462
   11       18013.3556857452      0.0010447397
   12       18013.2689253970      0.0009272101
   13       18013.1977138758      0.0008134316
   14       18013.1385397451      0.0007347676
   15       18013.0890506207      0.0006775458
   16       18013.0473047887      0.0006232717
   17       18013.0118160498      0.0005836780
   18       18012.9817069677      0.0005283921
   19       18012.9563792672      0.0005174258
   20       18012.9350679263      0.0004878701
   21       18012.9170103799      0.0004524497
   22       18012.9017521611      0.0004209502
   23       18012.8887621048      0.0004020105
   24       18012.8775780455      0.0003725634
   25       18012.8680448308      0.0003444192
```

```
26      18012.8598393359      0.0003335878
27      18012.8527732187      0.0003018124
28      18012.8466713236      0.0002872779
29      18012.8413712468      0.0002664523
30      18012.8367365647      0.0002504271
31      18012.8327273928      0.0002307838
32      18012.8292705867      0.0002235472
33      18012.8262661373      0.0002099719
34      18012.8236516003      0.0001995483
35      18012.8213494920      0.0001824696
36      18012.8193160973      0.0001727765
37      18012.8175257967      0.0001587713
38      18012.8159643619      0.0001516038
39      18012.8145778856      0.0001478936
40      18012.8133372617      0.0001356132
41      18012.8122184607      0.0001266768
42      18012.8112147532      0.0001182781
43      18012.8103143747      0.0001120542
44      18012.8095051710      0.0001036115
45      18012.8087901836      0.0000977979
46      18012.8081557185      0.0000971996
47      18012.8075824940      0.0000899283
48      18012.8070711273      0.0000861907
49      18012.8066086322      0.0000836632
```

The first column is the iteration number, the second is the function value at the corresponding iteration and the last column is the relative difference between subsequent iterations. The default stopping criteria of the function is that it either reaches 100 iterations or the relative difference in 5 subsequent iterations is no more than 10^{-4}. The denoised image can be seen in Figure 2

If we wish to solve the constrained problem (1.1) with $l = 0.2, u = 0.6$ (doesn't make much sense of course...), then one should write

```
>>X_den=denoise_bound(Bobs,0.02,0.2,0.6);
```

If the goal is to solve the denoising problem with nonnegativity constraints: $X_{ij} \geq 0$, then the appropriate command should be

```
>>X_den=denoise_bound(Bobs,0.02,0,Inf);
```

It is also possible to control different parameters of the algorithm by using a parameter structure. The parameters are

- **MAXITER** - maximum number of iterations. The default value is 100.

Figure 2: denoised camerman

- **epsilon** - tolerance parameter for the stopping criteria. The default value is 10^{-4}. If in 5 subsequent iteration the relative difference of the iterations is less than epsilon, the algorithm is terminated.
- **print** - If 1 then information on each iteration is given, otherwise, if 0, this information is suppressed.
- **tv** - type of tv regularizer. Either 'iso' for isotropic (default) or 'l1' for anisotropic regularizer.

Suppose that we wish to run the algorithm for no more than 10 iterations with l_1-based total variation regularizer. This is done by writing

```
clear pars
pars.MAXITER=10;
pars.tv='l1';
X_den=denoise_bound(Bobs,0.02,-Inf,Inf,pars);
```

2.2 Deblurring

Now we wish find a solution of problem (1.2) with $\mathcal{A}$ representing a blurring operation that we employed and $\mathbf{B}$ is the observed image `Bobs`. In this demonstration the regularization parameter λ is chosen to be 10^{-4}. We begin by defining a PSF array. For example, we can choose a uniform 3×3 blur:

```
>>P=1/9*ones(3,3);
>>center=[2,2];
```

The blurred and noisy image is constructed by the commands

```
>>randn('seed',314);
>>Bobs=imfilter(X,P,'symmetric')+1e-4*randn(size(X));
```

Note that here we used reflexive boundary conditions in the blurring operation. The function `deblur_tv_fista.m` solves problem (1.2) when the BC are either reflexive or periodic. In the case of reflexive BC, the PSF is assumed to be doubly symmetric (as is the case in the above example). The deblurring is made by writing

```
>>X_deblur=deblur_tv_fista(Bobs,P,center,0.001,-Inf,Inf);
```

The first argument is the observed image; the second and third arguments are the PSF and its center. The regularization parameter λ is the fourth argument. Finally, the fifth and sixth arguments are the lower and upper bounds on the pixels' values. In this example, the problem is unconstrained. The output is

```
*****************************
*   Solving with FISTA      **
*****************************
 #iter     fun-val     tv         denoise-iter      relative-dif
=======================================================================
  1       12.36849   1959.45247       10               0.01759
  2       9.92596    2057.89663       10               0.00903
  3       8.45815    2144.33237       10               0.00786
  4       7.53438    2216.88941       10               0.00691
  5       6.92154    2275.74416       10               0.00616
  6       6.49816    2322.18490       10               0.00556
  7       6.19813    2357.74397       10               0.00505
  8       5.98170    2383.70700       10               0.00460
  9       5.82386    2401.63869       10               0.00419
 10       5.70793    2413.35300       10               0.00381
  :          :           :             :                  :
 95       5.30148    2394.46241        6               0.00007
 96       5.30147    2394.45419        6               0.00007
 97       5.30147    2394.44667        6               0.00006
 98       5.30147    2394.43952        6               0.00006
 99       5.30147    2394.43348        6               0.00006
100       5.30147    2394.42835        6               0.00006
```

The first column is the iteration number, the second column is the function value at the current iteration. The third column is the value of the total variation, the fourth column is the number of denoising iterations performed at each deblurring step and the fifth column is the relative difference of the current and previous iterations. The output `X_deblur` is the deblurred image. The blurred and deblurred image are seen in Figure 3.

The default number of iterations is 100. If we want the function to execute a different number of iterations, say 20, we can change this parameter by defining a parameter structure

Figure 3: blurred and deblurred cameraman

```
>>clear pars
>>pars.MAXITER=20;
```

and then run the function with the `pars` argument

```
>>X_deblur=deblur_tv_fista(Bobs,P,center,0.001,-Inf,Inf,pars);
```

It is also possible to obtain an array containing all function values by adding a second output argument:

```
>>[X_deblur,fun]=deblur_tv_fista(Bobs,P,center,0.001,-Inf,Inf,pars);
```

There are many parameters that can be manipulated through the parameter structure (see also `'help deblur_tv_fista_sep'`):

- **MAXITER** - number of iterations of the FISTA method (default=100).
- **fig** - when this parameter is set to 1 (default), the image at each iteration is shown in Figure 314. If the value is 0, the visualization is suppressed.
- **BC** - boundary conditions. Either 'reflexive' (default) or 'periodic'. If the boundary conditions are reflexive, then the PSF is assumed to be doubly symmetric.
- **tv** - type of total variation regularizer. Either 'iso' for isotropic (default) or 'l1' for anisotropic.
- **mon** - when set to 1 (default) the monotone version of FISTA, namely MFISTA, is used. Otherwise, when mon=0, the "standard" nonmonotone version of FISTA is used.
- **denoiseiter** - maximum number of denoising iterations used at each iteration of the method (the actual number might be smaller if a stopping criteria is reached). Default is 10.

For example, if we want to invoke 13 iterations of the nonmonotone version of FISTA with at most 5 denoising steps at each iteration and with the l_1-based total variation regularizer, we can write

```
>>clear pars
>>pars.mon=0;
>>pars.MAXITER=13;
>>pars.tv='l1';
>>pars.denoiseiter=5;
>>X_deblur=deblur_tv_fista(Bobs,P,center,0.001,-Inf,Inf,pars);
```

Choosing the wrong BC might produce very poor quality reconstructions. For instance, the current example can be solved using periodic BC, although it was constructed assuming reflexive BC:

Figure 4: deblurring using periodic boundary conditions

```
>>clear pars
>>pars.BC='periodic'
>>pars.MAXITER=20;
>>X_deblur=deblur_tv_fista(Bobs,P,center,0.001,-Inf,Inf,pars);
```

The result is shown in Figure 4.

Another deblurring function in the package is `deblur_tv_fista_sep.m` which has the exact same interface as `deblur_tv_fista.m`. This function is dedicated to separable PSFs. Therefore, the second argument in the input of `deblur_tv_fista_sep.m` should be a separable PSF. If it is not, the function will use a rank one approximation. An important difference between the separable and non-separable functions is that in the former it is also possible to use zero boundary conditions and that in the reflexive BC case, the PSF does not necessarily has to be doubly symmetric.

References

[1] P. C. Hansen, J. G. Nagy, and D. P. O'Leary. *Deblurring images*, volume 3 of *Fundamentals of Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006. Matrices, spectra, and filtering.