Guide to the MATLAB code for wavelet-based deblurring with FISTA

Amir Beck and Marc Teboulle

October 23, 2008

1 General Description

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

$$\min_{\mathbf{X}} \|\mathcal{A}(\mathbf{X}) - \mathbf{B}\|^2 + \lambda \|\mathcal{W}(\mathbf{X})\|_1, \tag{1.1}$$

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.
- $\mathcal{W} : \mathbb{R}^{m \times n} \to \mathbb{R}^{m \times n}$ is an orthogonal linear transformation (possibly an orthogonal wavelet).
- λ - positive regularization parameter.

The functions are based on the paper

A. Beck and Marc Teboulle, "A Fast Iterative Shrinkage-Threshold Algorithm for Linear Inverse 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 first function in the package is

`deblur_wavelet_FISTA_trans.m`

which treats problems with either reflexive or periodic BC. In the reflexive BC case the PSF is assumed to be doubly symmetric. The second function in the package is

`deblur_wavelet_FISTA_sep.m`

which treats problems with separable PSF. In this case, in addition to periodic and reflexive BC, zeros BCS can also be handled. As opposed to the previous function, there is no restriction on the structure of the PSF in the reflexive BC case.

2 Examples

We begin by generating an artificial blurred and 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. Suppose now that the image is blurred with a Gaussian PSF generated by

```
>>[P,center]=psfGauss([9,9],4);
```

To actually generate the blurred image we write

```
>>B=imfilter(X,P,'symmetric');
```

The observed image is created by adding a noise to the blurred image. For example, we can add a Gaussian white noise with standard deviation `1e-3` to the image by

```
>>randn('seed',314);
>>Bobs=B +1e-3*randn(size(B));
```

To see the original and blurred/noisy images we can write

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

The result can be seen in Figure 1.

Now we wish find a solution of problem (1.1) with $\mathcal{A}$ representing the blurring operation that we employed and $\mathbf{B}$ is the observed image `Bobs`. In this demonstration the regularization parameter λ is chosen to be `1e-4`. Just for the illustration let us choose the orthogonal transformation as the two dimensional discrete cosine transform (Of course this will not produce a high quality restored image but the purpose here is to demonstrate the interface of the functions). The deblurring is done by writing

```
>>[X_out,fun_all]=deblur_wavelet_FISTA_trans(Bobs,P,center,@dct2,@idct2,1e-4);
```

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

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

and then run the function with the `pars` argument

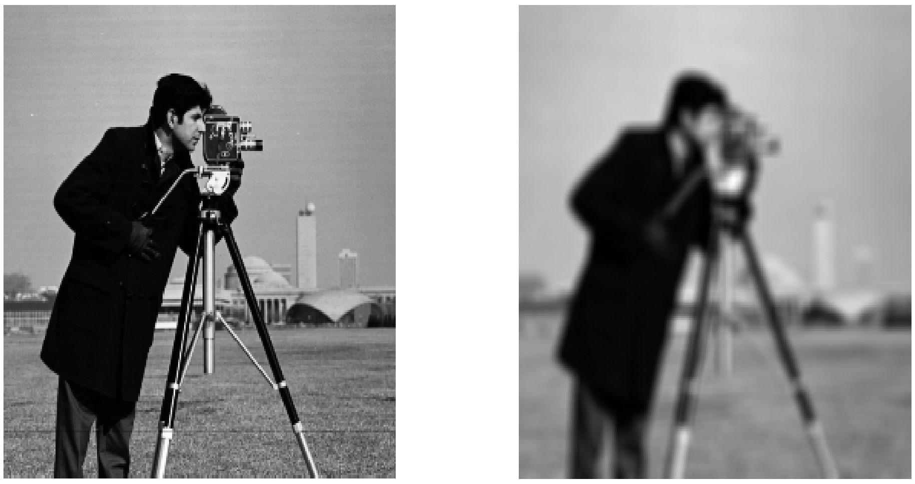

Figure 1: Blurring of the Cameraman

```
>>[X_out,fun_all]=deblur_wavelet_FISTA_trans(Bobs,P,center,@dct2,@idct2,1e-4,pars);
```

The output in this case is

```
*********
**FISTA**
*********

#iter          fun-val             relative-dif
====================================================
  1           10.50852                0.02317
  2            6.78424                0.01145
  3            4.76876                0.00947
  4            3.63094                0.00791
  5            2.93936                0.00676
  6            2.48239                0.00594
  7            2.15587                0.00535
  8            1.90727                0.00493
  9            1.70871                0.00462
 10            1.54432                0.00438
 11            1.40451                0.00420
 12            1.28330                0.00406
 13            1.17684                0.00394
 14            1.08259                0.00383
 15            0.99877                0.00374
 16            0.92404                0.00365
 17            0.85738                0.00356
 18            0.79789                0.00347
 19            0.74486                0.00339
 20            0.69764                0.00330
```

The first column contains the iteration number, the second column is the function value and the third column is the relative difference $\frac{\|\mathbf{X}_k-\mathbf{X}_{k-1}\|_F}{\|\mathbf{X}_{k-1}\|_F}$. Note that the result at each iteration is shown at figure 314. To supress the display of image at each iteration we can add an additional field to `pars`:

```
 >>pars.fig=0;
```

We will now use an orthogonal wavelet transform as a regularizer. We have chosen to use the wavelet function available at

`http://www.mathworks.com/matlabcentral/files/11133/wavelet.m`

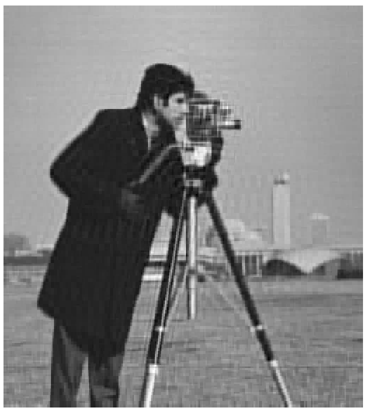

Figure 2: Deblurring of the Cameraman

that has a nicer interface than the one provided by the functions of the MATLAB wavelet toolbox. To operate the deblurring function with a two-stage Haar wavelet transform we can write

```
>>wav=@(X)wavelet('2D Haar',2,X);
>>iwav=@(X)wavelet('2D Haar',-2,X);
>>[X_out,fun_all]=deblur_wavelet_FISTA_trans(Bobs,P,center,wav,iwav,1e-4);
```

The resulting image can be seen in Figure 2.

The PSF that we have chosen in this example is in fact separable. Indeed, writing

```
>> rank(P)
```

produces the answer 1. Therefore, in this case the deblurring function that treats separable PSFs can also be used and will give exactly the same results:

```
[X_out,fun_all]=deblur_wavelet_FISTA_sep(Bobs,P,center,wav,iwav,1e-4);
```

Note that the BC can have a strong influence on the quality of the deblurred image. For example, if we use periodic boundary conditions, then we can run the following commands.

```
>>clear pars
>>pars.BC='periodic';
>>[X_out,fun_all]=deblur_wavelet_FISTA_sep(Bobs,P,center,wav,iwav,1e-4,pars);
```

The result appears in Figure 3 and obviously suffers from severe ringing effects.

We can also try to deblur it with zero boundary conditions.

```
>>pars.BC='zero';
>>[X_out,fun_all]=deblur_wavelet_FISTA_sep(Bobs,P,center,wav,iwav,1e-4,pars);
```

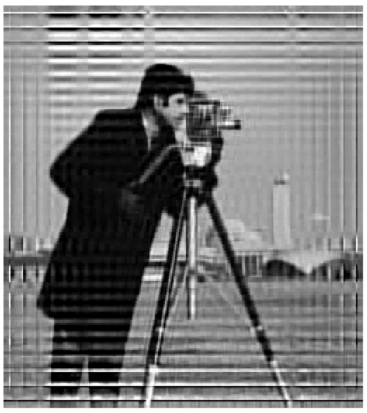

Figure 3: Deblurring of the Cameraman under periodic BC

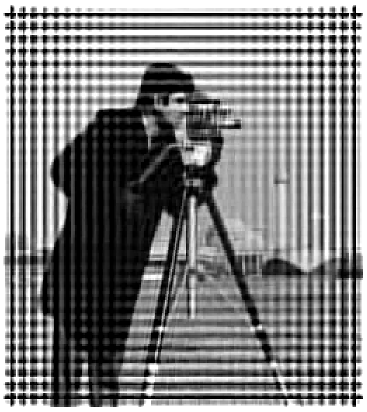

Figure 4: Deblurring of the Cameraman under zero BC

and the poor results can be seen in Figure 4.

Note that it is impossible to use zero boundary conditions in the function `deblur_wavelet_FISTA_tr`
Indeed, writing

```
>> [X_out,fun_all]=deblur_wavelet_FISTA_trans(Bobs,P,center,wav,iwav,1e-4,pars);
```

produces the following error message

```
 ??? Error using ==>deblur_wavelet_FISTA_trans at 100
  Invalid boundary conditions should be reflexive or periodic
```

3 m-files

3.1 The function deblur_wavelet_FISTA_trans

```
function[X_out,fun_all]=deblur_wavelet_FISTA_trans(Bobs,P,center,W,WT,lambda,pars)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function implements FISTA for solving the linear inverse problem with
% an orthogonal l1 wavelet regularizer and either reflexive or periodic boundary
% conditions
%
% Based on the paper
% Amir Beck and Marc Teboulle, "A Fast Iterative Shrinkage-Threshold Algorithm
% for Linear Inverse Problems", to appear in SIAM Journal on Imaging Sciences
% -----------------------------------------------------------------------
% Copyright (2008): Amir Beck and Marc Teboulle
%
% FISTA is distributed under the terms of
% the GNU General Public License 2.0.
%
% Permission to use, copy, modify, and distribute this software for
% any purpose without fee is hereby granted, provided that this entire
% notice is included in all copies of any software which is or includes
% a copy or modification of this software and in all copies of the
% supporting documentation for such software.
% This software is being provided "as is", without any express or
% implied warranty.  In particular, the authors do not make any
% representation or warranty of any kind concerning the merchantability
% of this software or its fitness for any particular purpose."
% ----------------------------------------------------------------------
%
% INPUT
```

```
%
% Bobs............................ The observed image which is blurred and noisy
% P .............................. PSF of the blurring operator
% center ..........................  A vector of length 2 containing the center
%                                           of the PSF
% W ..............................  A function handle. For an image
%                                   X, W(X)  is  an orthogonal
%                                   transformation of the image X.
% WT .............................  A function handle. For an image
%                                   X, WT(X) is the inverse
%                                   (with respect to the operator W)
%                                   orthogonal transform of the image X
% lambda .........................  Regularization parameter
% pars............................  Parameters structure
% pars.MAXITER ...................... maximum number of iterations (Default=100)
% pars.fig ...........................  1 if the image is shown at each iteration,
%                                         0 otherwise (Default=1)
% pars.BC ........................... boundary conditions.
%                                       'reflexive' (default) or 'periodic
% OUTPUT
%
% X_out .......................... Solution of the problem
%                                   min{||A(X)-Bobs||^2+lambda \|Wx\|_1
% fun_all ........................ Array containing all function values
%                                  obtained during the FISTA method


% Assigning parameters according to pars and/or default values
flag=exist('pars'); if (flag&isfield(pars,'MAXITER'))
    MAXITER=pars.MAXITER;
else
    MAXITER=100;
end if(flag&isfield(pars,'fig'))
    fig=pars.fig;
else
    fig=1;
end if (flag&isfield(pars,'BC'))
    BC=pars.BC;
else
    BC='reflexive';
end if (flag&isfield(pars,'tv'))
    tv=pars.tv;
else
```

```
    tv='iso';
end

% If there are two output arguments, initalize the function values vector.
if (nargout==2)
    fun_all=[];
end

[m,n]=size(Bobs); Pbig=padPSF(P,[m,n]);

switch BC
    case 'reflexive'
        trans=@(X)dct2(X);
        itrans=@(X)idct2(X);
        % computng the eigenvalues of the blurring matrix
        e1=zeros(m,n);
        e1(1,1)=1;
        Sbig=dct2(dctshift(Pbig,center))./dct2(e1);
    case 'periodic'
        trans=@(X) 1/sqrt(m*n)*fft2(X);
        itrans=@(X) sqrt(m*n)*ifft2(X);
        % computng the eigenvalues of the blurring matrix
        Sbig=fft2(circshift(Pbig,1-center));
    otherwise
        error('Invalid boundary conditions should be reflexive or periodic');
end
% computing the two dimensional transform of Bobs
Btrans=trans(Bobs);

%The Lipschitz constant of the gradient of ||A(X)-Bobs||^2
L=2*max(max(abs(Sbig).^2));

% initialization
X_iter=Bobs; Y=X_iter; t_new=1; fprintf('************\n');
fprintf('**FISTA**\n'); fprintf('************\n'); fprintf('#iter
fun-val         relative-dif\n==============================\n');
for i=1:MAXITER
    % Store the old value of the iterate and the t-constant
   X_old=X_iter;
   t_old=t_new;

    % Gradient step
    D=Sbig.*trans(Y)-Btrans;
    Y=Y-2/L*itrans(conj(Sbig).*D);
    Y=real(Y);

    % Wavelet transform
    WY=W(Y);
    % Soft thresholding
    D=abs(WY)-lambda/(L);
    WY=sign(WY).*((D>0).*D);
    % The new iterate inverse wavelet transform of WY
    X_iter=WT(WY);


    %updating t and Y
    t_new=(1+sqrt(1+4*t_old^2))/2;
    Y=X_iter+(t_old-1)/t_new*(X_iter-X_old);

    % Compute the l1 norm of the wavelet transform and the function value
    % and store it in the function values vector fun_all if exists.
    t=sum(sum(abs(W(X_iter))));
    fun_val=norm(Sbig.*trans(X_iter)-Btrans,'fro')^2+lambda*t;
    if (nargout==2)
        fun_all=[fun_all;fun_val];
    end
    % printing the information of the current iteration
    fprintf('%3d    %15.5f                %15.5f \n',
    i,fun_val,norm(X_iter-X_old,'fro')/norm(X_old,'fro'));

    if (fig)
        figure(314)
        imshow(X_iter,[])
    end
end

X_out=X_iter;
```

3.2 The function deblur_wavelet_FISTA_sep

```
function
[X_out,fun_all]=deblur_wavelet_FISTA_sep(Bobs,P,center,W,WT,lambda,pars)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function implements FISTA for solving the linear inverse problem with
% an orthogonal l1 wavelet regularizer and a seperable PSF
%
```

```
% Based on the paper
% Amir Beck and Marc Teboulle, "A Fast Iterative Shrinkage-Threshold Algorithm
% for Linear Inverse Problems",  to appear in SIAM Journal on Imaging Sciences
% -----------------------------------------------------------------------------
% Copyright (2008): Amir Beck and Marc Teboulle
%
% FISTA is distributed under the terms of
% the GNU General Public License 2.0.
%
% Permission to use, copy, modify, and distribute this software for
% any purpose without fee is hereby granted, provided that this entire
% notice is included in all copies of any software which is or includes
% a copy or modification of this software and in all copies of the
% supporting documentation for such software.
% This software is being provided "as is", without any express or
% implied warranty.  In particular, the authors do not make any
% representation or warranty of any kind concerning the merchantability
% of this software or its fitness for any particular purpose."
% ---------------------------------------------------------------------------
%
% INPUT
%
% Bobs............................ The observed image which is blurred and noisy
% P .............................. PSF of the blurring operator
% center .........................  A vector of length 2 containing the center
%                                   of the PSF
% W ..............................  A function handle. For an image
%                                   X, W(X)  is  an orthogonal
%                                   transformation of the image X.
% WT .............................  A function handle. For an image
%                                   X, WT(X) is the inverse
%                                  (with respect to the operator W)
%                                   orthogonal transform of the image X
% lambda .........................  Regularization parameter
% pars............................  Parameters structure
% pars.MAXITER ...................... maximum number of iterations
%                                     (Default=100)
% pars.fig .......................... 1 if the image is shown at each
%                                     iteration, 0 otherwise (Default=1)
% pars.BC ........................... boundary conditions.
%                                     'reflexive' (default) , 'periodic or 'zero'
% OUTPUT
%
```

```
% X_out ........................... Solution of the problem
%                                    min{||A(X)-Bobs||^2+lambda \|Wx\|_1
% fun_all ......................... Array containing all function values
%                                   obtained during the FISTA method


% Assigning parameters according to pars and/or default values
flag=exist('pars'); if (flag&isfield(pars,'MAXITER'))
    MAXITER=pars.MAXITER;
else
    MAXITER=100;
end if(flag&isfield(pars,'fig'))
    fig=pars.fig;
else
    fig=1;
end if (flag&isfield(pars,'BC'))
    BC=pars.BC;
else
    BC='reflexive';
end if (flag&isfield(pars,'tv'))
    tv=pars.tv;
else
    tv='iso';
end

% If there are two output arguments, initalize the function values vector.
if (nargout==2)
    fun_all=[];
end

[m,n]=size(Bobs); Pbig=padPSF(P,[m,n]);

[m,n]=size(Bobs); Pbig=padPSF(P,[m,n]);
% computing the terms of the kronecker product
[Ar,Ac]=kronDecomp(Pbig,center,BC);
% computing the singular values of the Kronecker product of Ar and Ac
sr=svd(Ar); sc=svd(Ac); Sbig=sc*sr';

%The Lipschitz constant of the gradient of ||A(X)-Bobs||^2
L=2*max(max(abs(Sbig).^2));

% initialization
X_iter=Bobs; Y=X_iter; t_new=1; fprintf('***********\n');
```

```
fprintf('**FISTA**\n'); fprintf('***********\n'); fprintf('#iter
fun-val         relative-dif\n===============================\n');
for i=1:MAXITER
    % Store the old value of the iterate and the t-constant
   X_old=X_iter;
   t_old=t_new;

    % Gradient step
     Y=Y-2/L*Ac'*(Ac*Y*Ar'-Bobs)*Ar;

    % Wavelet transform
    WY=W(Y);
    % Soft thresholding
    D=abs(WY)-lambda/(L);
    WY=sign(WY).*((D>0).*D);
    % The new iterate inverse wavelet transform of WY
    X_iter=WT(WY);


    %updating t and Y
    t_new=(1+sqrt(1+4*t_old^2))/2;
    Y=X_iter+(t_old-1)/t_new*(X_iter-X_old);

    % Compute the l1 norm of the wavelet transform and the function value
    % and store it in the function values vector fun_all if exists.
    t=sum(sum(abs(W(X_iter))));
    fun_val=norm(Ac*X_iter*Ar'-Bobs,'fro')^2+lambda*t;
    if (nargout==2)
        fun_all=[fun_all;fun_val];
    end
    % printing the information of the current iteration
    fprintf('%3d    %15.5f                %15.5f \n',
    i,fun_val,norm(X_iter-X_old,'fro')/norm(X_old,'fro'));

    if (fig)
        figure(314)
        imshow(X_iter,[])
    end
end

X_out=X_iter;
```

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.