Sparse Representations and Their Applications in Signal and Image Processing

Home Work 2: Iterative Shrinkage Algorithms

Submitted by:

First Student	1234567890
Second Student	0987654321

due 4.1.2016

Instructions

- Work should be done in pairs and submitted via moodle. Students wishing to submit on their own need to get permission to do so in advance.
- Submission should include a pdf-file containing the report and all Matlab files you created for this exercise.
- You are encouraged (but not obligated) to use LaTeX for your report, and add your answers to the project `.tex` file that is included with the exercise files. *You will get a 5 point bonus for doing so. However, the maximal grade will be 100 in any case for the exercise.*
- The items that require you to code or write an answer have an icon of keyboard or pen respectively.

  If you decide to submit your file in LaTeX, make sure your answers and figures are placed inside a `\BeginAnswer` ...`\EndAnswer` block that places what you added in a box such as this one.

- Write your own code. This exercise is meant to teach you how the different algorithms operate.
- Indent and highlight the Matlab code you attach to the report. In LaTeX this is done using the command: `\lstinputlisting{<filename>}` from the package `mcode.sty`, which is already included in the homework laTeX file.
- Use sparse type vectors and matrices in your code when it is appropriate.
- Questions regarding the exercise should be asked in the course forum in moodle, e-mails about the exercises will not be answered.

Part A- Synthetic Data Generation

The goal of this exercise is to get you acquainted with the various algorithms and to measure their performance. Throughout the assignment, you will be seeking to minimize the following expression:

$$f(x) = \frac{1}{2}\|y - CDx\|_2^2 + \lambda\|x\|_1 \tag{1}$$

Here y represents an image and x is its representation using the dictionary D. Our dictionary is going to be the 2-D Haar basis (inverse of the 2-D Haar transform, which is simply the transpose of the Haar transform as the Haar wavelet is unitary). We will use the following convention: Y_0 is an $s \times s$ image we wish to recover and $y_0 = \text{vec}(Y_0)$ is its column stack representation. The 2-D Harr transform coefficients are $X = \text{haar2D}(Y_0)$ ($s \times s$ matrix), and $x = \text{vec}(X)$ is its column stack vector. We are presented with a corrupted measurement y, which is related to x using the relationship $y = CDx + n$, where n is a noise vector, D represents the inverse of the Haar transform and C is a sampling matrix.

1. Dictionary - D

- Set $s = 128$ and define D to be the $128^2 \times 128^2$ matrix representing the 2-D Haar basis in *vector* form (hence the s^2 size). *At this point you need to start using the* `sparse` *matlab function*

2. Sampling Matrix - C

We will be experimenting with three types of sampling matrices, defined using a parameter p that stands for the 'compression' or amount of omitted pixels. The size is going to be $m \times n$ where $n = s^2$ is the dimension of the unknown signal x (remember D is square) and $m = \lfloor ps^2 \rfloor$ is the amount of measurements we receive.

1. C_d of size $m \times n = \lfloor ps^2 \rfloor \times s^2$ represents deletion of pixels from an image. Generate this matrix by deleting $n - m$ random rows from the identity matrix of size $s^2 \times s^2$, and then multiply it by an averaging filter of size l, e.g. as follows:

```
l = 2;
v = [ones(1,l) / l, zeros(1,s-l)];
M = sparse(toeplitz(v,v));
C_a = kron(M,M);
C_d = eye(n);
C_d = C_d(sort(randperm(n,m)),:);
C_d = sparse(C_d*C_a);
```

After C_d is generated check to see that none of the atoms in D are zeroed entirely when multiplied by C_d. if that happens, regenerate the matrix, if the problem persists enlarge l.

2. C_g is a Gaussian sampling matrix (known also as a sensing matrix) of size $m \times n = \lfloor ps^2 \rfloor \times s^2$, with i.i.d. entries drawn uniformly at random from $\mathcal{N}(0,1)$

3. C_b of size $n \times n$, which represents a 11-tap Gaussian blur kernel. Use the following code snippet to generate it:

```
%Half of an 11 tap Gaussian filter:
v = exp(-0.5*(0:5).^2);
%Normalize:
v = v/(v(1) + 2*sum(v(2:end)));
%Create the 1-D filter matrix of size sxs:
M = sparse(toeplitz([v, zeros(1,s-numel(v))],[v, zeros(1,s-numel(v))]));
%Create the 2-D vector form blur operator:
C_b = kron(M,M);
```

Write your code in such a way that the multiplication CD is never calculated nor stored, i.e. your code contains only vector and matrix operations.

3. Measurement y and the Unknown x

- Generate the set T of size k (e.g. $k = 30$) from the set $\{1 \ldots n\}$ (without repetitions). This is going to be the 'true' support. Use the following code snippet:

```
T = randperm((s/4)*(s/4+1),min(ceil(k/2),40));
T = mod(T,s/4) + s*(floor(T/(s/4)));
while length(T) < k
T = unique([T, randperm(n,k-length(T))]);
end
```

- Generate x, the unknown parameter, such that $x_i \sim \mathcal{N}(0,1),\ i \in T$; $x_i = 0,\ i \notin T$.

- Generate $y_0 = Dx$ that represents the clean, uncompressed signal.

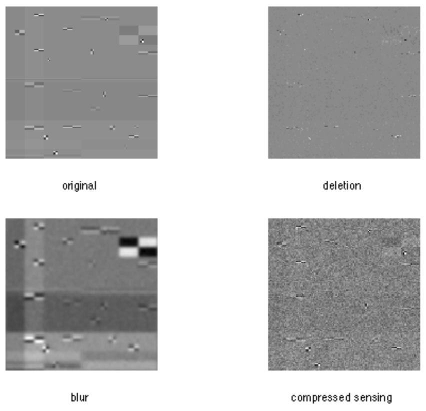
- Generate $y = Cy_0 + w$ where w is an i.i.d. Gaussian white noise vector of appropriate length with variance σ_n^2. This time set σ_n `(=sig_n)=dr_ratio*dr`, where $dr = max(Cy_0) - min(Cy_0)$, and `dr_ratio` is an input to the function `GenData` (see below).

4. Wrapping Function

⌨ Write the function
`[C_g, C_d, C_b, D, T, x, y_d, y_g, y_b, y0]=GenData(p,k,dr_ratio).`

- Below you can see an example for `p`=20% and `dr_ratio` = 2%. The figures are the matrix form of the vectors:

- original: y_0
- deletion: $C_d^T y_d = C_d^T (C_d y_0 + w)$
- blur: $y_b = C_b y_0 + w$
- compressed sensing: $C_g^\dagger y_g = C_g^\dagger (C_g y_0 + w)$.



Part B- Initial Results

Let us compare some straight-forward solutions to our problem of recovering x from y in (1). Our tools for comparison will be the following:

1. $PSNR(y_0, \hat{y}) = db(s^2 \cdot dr^2 / \|y_0 - \hat{y}\|_2^2)$, where $dr = max(y_0) - min(y_0)$ is the dynamic range, and $db(\cdot) = 10 \log_{10}(\cdot)$.

2. $SSIM(Y_0, \hat{Y})$ - Structured Similarity Index Measure, can be calculated by the matlab function of the same name.

3. $MSE(x, \hat{x}) = \|x - \hat{x}\|_2^2 / \|x\|_2^2$ - Mean Squared Error in x.

Above, $\hat{Y}$ represents the reconstructed image of size $s \times s$ and $\hat{x}$ is its estimated coefficients. More specifically $\hat{Y}$ is $\hat{y}$ rearranged as an image and $\hat{y} = D\hat{x}$
.

Use the following estimators for $\hat{y}$:

1. Oracle - Calculate the image that minimizes $\|y - CDx\|_2$ given that the **true support T is known**. *In reality we don't have access to this information, but under certain conditions some methods can achieve 'oracle' or 'near oracle' performance making the oracle estimator a good tool for comparison*

2. L_2-estimator - compute the image that solves the following optimization problem:
$$\hat{y} = D\hat{x}, \qquad \hat{x} = \min_{\hat{x}} \|\hat{x}\|_2^2 \;\; s.t. \;\; y = CD\hat{x}$$

Do this by running the conjugate gradient method (google it) for at least 500 iterations, and remember to use only vector-matrix multiplications (i.e. no multiplying C and D directly)

- ⌨ Write the file `PartB.m` that evaluates the performance of the oracle and L_2 estimators. Present the results in a clear way and calculate the $PSNR$, $SSIM$ and MSE of each method. Present the resulting images alongside the original y_0 you generated. **Pick a value (as you wish) for `dr_ratio` from the range 1%-3%, for `p` from the range 10%-30% and for `k` from the range 30-60. Do not change these values for the remainder of the exercise (bar the last part)**
- ✏ Discuss the results and try to explain the performance you get.

Part C- Shrinkage Algorithms

1. Implementation

Implement the following algorithms. You might need to visit Wikipedia and chapter 6 in the course book [2] (or use other sources if you wish) to learn more on some of the subjects mentioned.

- ⌨ `ista` - Iterative Shrinkage Thresholding Algorithm [2, fig. 6.1].
- ⌨ `ista_ls` - ISTA with a a line search - [2, p. 127], using the armijo rule (Google it).
- ⌨ `fista` - FISTA with a constant step size [1, section 4]

Implementation notes:

- Initialize all algorithms with $\hat{x} = 0$.
- Use the following f:

```
f = @(x) lambda * norm(x,1) + 0.5 * norm(y - C*(D*x),2)^2
```
(2)

- Calculate and store the value of f and the PSNR at each iteration of the algorithms. Plot these quantities (f and PSNR as function of the iteration number) along with the algorithms' final output.
- There is no need to normalize the columns of CD to unit norm. You can do this if you want but if you decide to do that be careful to do it efficiently.
- The algorithms should get both C and D as their input.

- You should implement two stopping criteria for your algorithms: (i) difference between subsequent estimates is small (i.e., $\|\hat{x}_{i+1} - \hat{x}_i\|_2^2 \leq \epsilon_t$ for some small constant ϵ_t, where i is the iteration number); and (ii) a limit on the maximal amount of iterations the algorithm can run. The algorithm should stop if one of these criteria is met.

- *Advice:* when debugging your implementation, you might want to set s to be smaller than 128, so the code will run faster.

2. Testing

⌨ Create the file `PartC.m` to hold the code for the tests you perform in this section.

- Work in this fashion:

  1. Find the maximal eigen-value ($\lambda_{\max}$, a.k.a. the matrix norm) of $D^T C^T C D$ using the power method (google it), and set $c = \lambda_{\max}$ in ISTA.
  2. Use ISTA to seek for a good value of λ that yields a good PSNR and a sufficiently sparse solution (with less than $10 \cdot k$ non-zeros). The limit of the number of ISTA iterations in this step for every candidate value of λ should be at least 1000.
  3. Run all three algorithms with the λ you found until convergence is reached or the limit on the number of iterations is met.
  4. Use debiasing after you get a solution, similar to the LARS part of the previous homework assignment. You may want to use the conjugate gradient algorithm to solve the least squares problem you get in the debiasing step.

- Plot a graph of the value of f and the PSNR as a function of the iterations for each of the methods and the three types of sensing matrices.

✎ Is the PSNR monotonically decreasing? *Hint: there are papers that deal with this issue, e.g. [3].*

⌨ Compare the resulting SSIM, MSE, PSNR, and the sparsity of the obtained solution for each of the methods. Present the resulting images.

✎ Compare the MSE and PSNR before and after debiasing, did it improve? by how much?

✎ Discuss the results: Which method is better and when? What are your conclusions?

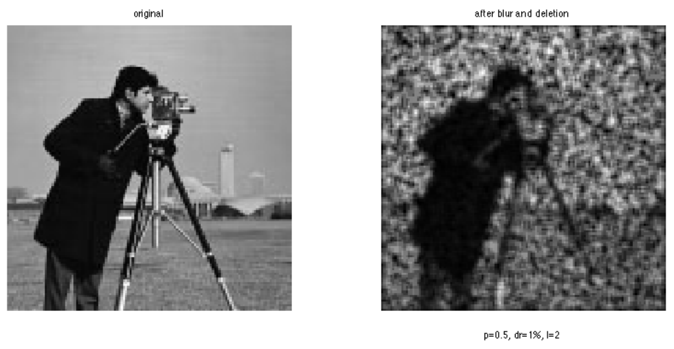
Part D- De-Blurring and Inpainting of a Real World Image

We will now try to de-blur and inpaint a real world image. Often this is done using redundent dictionaries (unlike the square orthogonal dictionary used in this exercise). An example for such a redundent dictionary is the undecimated Haar transform (see for example the deblurring examples in [1] and [3] for more details). Here, we will settle with the orthogonal Haar that you have already generated.

- Create the file `PartD.m`
- Set:

  ```
  Y0 = double(rgb2gray(imread('cameraman.jpg')));
  Y0 = Y0 / max(Y0(:));
  ```

  The file `cameraman.jpg` is supplied with the exercise files.
- Plot the energy in the coefficients x_0 sorted in descending order.
- Is the sparsity assumption valid for this image? Explain.
- Repeat the experiments in Part B. using only C_d (dont forget to apply the averaging filter before deleting pixes, as explained above) with $p = 0.5$ and $dr = 1\%$. For the oracle estimator support choose a few thousands of the largest coefficients (in absolute value) of x_0 in the Haar basis. See the following figure for how the original and the generated image should look like:



- Now run the algorithms from Part C. Try to achieve a solution with $\sim 4000 - 5000$ non zeros
- Explain the role of λ in the output result. What will happen if λ is too small? What will happen if it is too large?

✎ Did debiasing help in this case?

✎ Present and discuss the results you get.

⌨ Bonus challenge: try to de-blur an image of your own (of size $128\times$ 128).

Wrap Up

⌨ Create the file `RunMe.m` that can be used to run all the code for this exercise and recreate your results

- Submission in moodle should include two files:
  - Your report in PDF format, named `SparseRepHw2_<student #1>_<student #2>.pdf`.
  - Compressed ZIP file named `SparseRepHw2_<student #1>_<student #2>.zip` containing your report and matlab code.

References

[1] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[2] Michael Elad. *Sparse and Redundant Representations*. Springer-Verlag New York, 2010.

[3] Raja Giryes, Michael Elad, and Yonina C Eldar. The projected gsure for automatic parameter tuning in iterative shrinkage methods. *Applied and Computational Harmonic Analysis*, 30(3):407–422, 2011.