EL-GY 6123 Image and Video Processing, Fall 2017

Programming Assignment 2 (Due 2/27/2017)

1) Write a function implementing ISTA algorithm for solving the LASSO problem using Python.

Lasso problem: $J(\mathbf{x}) = \|\mathbf{y} - \mathbf{H}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1.$

ISTA algorithm: $\mathbf{x}_{k+1} = \text{soft}\left(\mathbf{x}_k + \frac{1}{\alpha}\mathbf{H}^\mathsf{T}(\mathbf{y} - \mathbf{H}\mathbf{x}_k),\ \frac{\lambda}{2\alpha}\right)$, $\alpha \geq \text{maxeig}(\mathbf{H}^\mathsf{T}\mathbf{H})$

The program should take the dictionary matrix **H** and signal vector **y** as input, as well as the parameter λ, and return the solution x. It should automatically determine the parameter using α=maxeig($\mathbf{H}^\mathsf{T}\mathbf{H}$). To find the eigenvalue of a matrix, you could use numpy.linalg.eig().(note you have to order the eigenvalues to find the maximum).
You can choose to start with any initial condition (e.g. x_0=0), and set the convergence condition as error_ratio=(old_error-new_error)/old_error< T (e.g. T=1e-), where the error refers to the residual L2 norm: $\|\mathbf{y}-\mathbf{H}\mathbf{x}\|_2$.

Hint: You can consult the MATLAB program in Ref [1]. However, instead of specifying the number of iterations, I would like you to use a convergence criterion that checks for error reduction ratio. Basically you program can have a while loop. Within each iteration, it updates **x** from the previous solution using the ISTA update rule. Afterwards you check the new error with the new solution, and compare it with the old error. If the error_ratio <T, then you exit the while loop; Otherwise, you set old_error=new_error, and old x=new x, and go to the next iteration.

Ref [1]: Ivan Selesnic, “Sparse Signal Recovery”, http://eeweb.poly.edu/iselesni/lecture_notes/sparse_signal_restoration.pdf

2. Test your function for a dimension N=4 vector with sparse representation using 1D DCT transform. You need to generate a dictionary **H** that includes all the 1D DCT basis vectors of a given dimension N, and a coefficient vector **x** that is relatively sparse (i.e., only a few coefficients are non-zero), and generate a data vector y using **y=Hx**. You can either use this **y** vector directly or perturb it slightly to simulate noise. You then run your program to see whether it provides correct solution **x** or how far it is from the real solution. To generate DCT basis, you should write your own function: DCT_basis_gen(N) which takes dimension N as input and returns 1D DCT basis vectors. You could use the following equation to generate your DCT basis:

$$\text{Basis Vectors}:$$

$$h_k(n) = \alpha(k)\cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

$$where\ \alpha(k) = \begin{cases} \sqrt{1/N} & k = 0 \\ \sqrt{2/N} & k = 1,\ldots,N-1 \end{cases}$$

Note that in the above notation, k is the index of the basis (the column index of the dictionary), and n is the index for the vector component. For example, to obtain the first DCT basis, you set k=0, and let n=0,1,2,3 to obtain the 4 components of the basis vector.

3. Using your ISTA function for image denoising in a block-by-block manner, using the 2D DCT basis as the dictionary **H** and a block size of 8x8. Your main program should read in an image, add random noise at a specified noise level, and run the ISTA algorithm for each 8x8 block (without overlapping) and replace the noisy block (or save it in a separate image) by the denoised block. Try two different noise levels (e.g. 0.01*255 and 0.1*255) and for each noise level, try two different values for λ (e.g. λ=1 and 10).

   Note that you need to convert each 8x8 block **y** into a vector, also determine the corresponding dictionary **H** from the DCT basis images. To determine all 1D DCT basis vectors, you could use the function DCT_basis_gen(N) from problem 1. Then you can form all 2D DCT basis images using the outer-product of 1D basis vectors. Finally you need to convert each 2D DCT basis image into a vector to form one of the dictionary atom. After you find the solution **x**, the denoised vector **Hx** needs to be reordered into a 2D block and put at the appropriate location of the denoised image. You could use numpy.reshape() to convert matrix to vector and vice versa.
   In order to reduce the computation time, you could choose to only work on a small portion of a large image and only the Y component if your image is a color image.

4. Perform Wavelet Based Denoising. In this case, instead of converting your entire image into a long vector, and form a dictionary from the wavelet transform basis images, I suggest that you modify your ISTA function, to directly work with 2D images. Note that **Hx** corresponds to performing inverse 2D transform, and $\mathbf{H}^T\mathbf{y}$ corresponds to forward transform. The remaining operation is element-wise operation. Therefore, you can rewrite your ISTA function to directly work with 2D images (i.e., y is input noisy image, and x are the wavelet transform coefficients represented in whatever data structure that is returned by the 2D wavelet transform, and you do not need to explicitly save the dictionary matrix **H**. Instead you can program **Hx** as a function inverse_transform(x), and $\mathbf{H}^T\mathbf{y}$ as another function forward_transform(y). The update of **x** can be done for each individual element in **x**.
   In Python, you can use pywt.wavedec( ) for forward transform, and pywt.waverec( ) for inverse transform. For this exercise, pick one noise level and λ, show results using two different wavelets (Haar wavelet and the Daubechies 8/8 wavelet). Use 3 level decomposition. (You are encouraged to compare results obtained with different levels of decomposition. But this is not necessary)

   Please include plots of original image, noisy image, the wavelet transform of the noisy image, the wavelet transform of the final denoised image (which is the solution of sparse coding), and the final denoised image. Please also include a plot of the error vs. iteration numbers. Please comment on the pros and cons of using different wavelet filters.