AMATH 582: ASSIGNMENT 5

JON STAGGS

Applied Mathematics M.S. Student, University of Washington, Seattle, WA
jrstaggs@uw.edu

Abstract. We are provided an image and we explore the role of sparsity in a particular basis and the compression of images. We also discuss the use of Lasso in the the sparse recovery problem of a corrupted images/image with limited measurements.

1. Introduction and Overview

We are provided an image and we analyze the image in the real part of the Fourier basis to explore it's sparsity in this basis which will allow us to compress the image. We also use convex optimization to recover a compressed image with varying amounts of measurements from the original signal. We build up the theoretical framework to perform the compression and recovery. Next we discuss the relevant Python packages and algorithm to computationally carry out this problem. We explore the sparsity in the Fourier Domain and carry out an image compression while retaining different amounts of coefficients. We also perform a recovery on an image that has been corrupted/has limited measurements from the original image. We use this on an unknown mystery image as well.

2. Theoretical Background

We can formulate the corrupted image problem as a signal recovery problem. We use the notation from [5]. We consider an image $f^\dagger : [0,1] \to \mathbb{R}$ and discrete measurements $\underline{y} \in \mathbb{R}^M$ where $\underline{y}_j = f^\dagger(t_j)$. We consider the problem where we have imperfect measurement - that is measurements with white noise. We can formulate this as $\underline{y^\dagger}_j = \xi_j f^\dagger(t_j)$ with $\xi_j \sim \mathcal{N}(0,1)$. We will discuss how we can exploit sparsity in a particular basis for compression of an image/signal as well as how to recover a compressed image using convex optimization.

2.1. Representation in the Fourier Basis.

Our goal is to recover an approximation $\hat{f}$ to our original image $f^\dagger$ given our noisy measurements $\underline{y}^\dagger$. We assume that $\hat{f}(t_j) = \sum_{m,n} \beta_{m,n}\psi_{m,n}(t_j)$ where $\psi_{m,n}$ are the real parts of the FFT [5],[4]. In the discrete setting, this is the Discrete Cosine Transform,

$$\text{DCT}(f)_k = \sqrt{\frac{1}{K}}\left[f_0 \cos\left(\frac{\pi k}{2K}\right) + \sqrt{2}\sum_{j=1}^{K-1} f_j \cos\left(\frac{\pi k(2j+1)}{2K}\right)\right]. \tag{1}$$

For more information on the Fourier Transform and its discrete analog see [9]. Need to say something about writing our image matrix as a vector We can now write our model in the typical $Ax = b$ form as,

$$W\underline{\beta} = \hat{f} \text{ with } W = \begin{bmatrix} \psi_0(t_0) & \psi_1(t_0) & \cdots & \psi_{J-1}(t_0) \\ \psi_0(t_1) & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \psi_0(t_{N-1}) & \cdots & \cdots & \psi_{J-1}(t_{N-1}) \end{bmatrix}. \tag{2}$$

Date: March 18, 2022.

Since we are interested in a noisy measurement vector we have the problem $\underline{y}^{\dagger} = A\beta$ where $A = BW$ with $B \in \mathbb{R}^{M\times N}$ and B has rows that correspond to vectors drawn from $\mathcal{N}(0,1)$.

2.1.1. *Recovery From Compressed Signal.* The above framework leads us to a supervised learning problem that we formulate into a convex optimization problem found in [4],

$$\text{minimize}_{x\in\mathbb{R}^{\mathbb{N}}}||\beta||_1 \text{ subject to } A\beta = y. \tag{3}$$

Here we make the remark that we define A so that we are implicitly constructing/finding a β that is in our Fourier basis.

2.2. **Sparsity and Lasso.** In eqn. (3) notice that we are minimizing with respect to the 1-norm. We will explore later, but the choice of using the DCT basis comes from the fact that our image will be sparse in this basis. The intuition for using the l_1 norm (for the author at least) is that the l_1 norm is the sum of the absolute value of the components. Since we are in a sparse setting then if we try to minimize the l_1 norm we will pick up lots of pieces that have small magnitude. The constraint that $A\beta = y$ keeps this in check so we'd expect the optimal β^* to pick up the most important values that can approximate y and try to grab the smallest components for everything else. Moreover, we are working with an underdetermined system which means there are infinite system and minimizing in l_1 as we just explained will tend to favor components that are zero [5].

3. Algorithm Implementation and Development

We use Python version 3.8 [10] with SciKit-learn [7] (for image rescaling), NumPy [2], SciPy [11], CVXPY [1], and MatplotLib [6] libraries and Sypder as our IDE [8]. We will briefly discuss the preprocessing and then describe the implementation.

Our original image has the dimensions 292×228 (in pixels). We use the *ski* package (from SciKit-learn) and the *rescale* method to rescale the dimensions of the image we consider. We rescale to 18% so that the dimension of the image we consider is $N_x \times N_y = 53 \times 41$. We construct the forward and inverse discrete transform as defined in [3] and denote them as D, D^{-1} respectively. These transform matrices are dimension $N_xNy \times N_xNy$ - for notational convenience we take $N = N_xN_y$. We also turn our image matrix into a vector, $\hat{f}$, of length 2173.

3.1. **CVXPY.** The CVXPY package is a modeling language for convex optimization problem [1]. We define our optimization variable with the *Variable* method whose argument is a specified dimension. Our objective function is defined by the quantity that we are attempting to minimize. We use the *Minimize* method and give it the argument of the one norm of our variable. The *Problem* class which takes in the objective and constraint as arguments. The *Problem* class is how CVXPY specifies the optimization problem that we are solving. The *solve* method numerically solves the optimization problem specified by the *Problem* class. We then use the *value* method on our variable to obtain the optimized x being x^* (or β, β^*) to use the notation from Sec. 3.2.

3.2. **Image Compression Algorithm.** We compute the DCT of our image vector to obtain the DCT coefficients. To carry out the image compression we then sort the coefficients of the DCT of our image vector and use Numpy's *percentile* method which computes the value of a specified percentile. For the top P percentile we take all values above the returned value from Numpy's *percentile*. We then use Numpy's *where* method to obtain the indices of the coefficients corresponding to the top P percentile. Next we create a zero vector and place the top P percentile coefficients into the corresponding index positions. This is our compressed DCT coefficient vector. To reconstruct the image we simply build our inverse DCT matrix and left multiply our vector by the inverses transform matrix. We then reshape this vector back into a matrix with original dimensions.

FIGURE 1. Downsized Gray Scale Version of "Son of Man" image provided.

3.3. **Compressed Image Recovery Algorithm.** We first build our measurement matrix, B, with M measurements. This corresponds to taking an $N \times N$ identity matrix and permuting M rows by selecting M random rows from length N and substituting these rows in an identity matrix. Our resulting measurement matrix has dimensions $M \times N$. We also construct the DCT and Inverse DCT matrices of length $N \times N$. We form y by taking $B\hat{f} = y$ which represents our measurements of the image vector. We construct A by taking BD^{-1} and this is our measurement matrix in our original basis. We then form our optimization problem defined in eqn. (3). We define our optimization variable x with dimension $N = N_x N_y$ (being a vector version of our matrix). Again note that how we construct/define A and y that we are searching for an x in our Fourier/DCT basis. We then define our objective with *cp.Minimize(cp.norm(x,1))* where the 1 denotes the 1 norm which allows us to exploit the sparsity in the Fourier basis. We then define the equality constraint $Ax = y$. Using the *Problem* we input our objective and constraint to initialize our optimization problem in CVXPY. Using the *solve* method we find a numerical solution to the convex optimization problem. We now have our x^* which [4] points out is the DCT of a vector F^* which should be an approximation to our original image vector F. All that remains is to reshape this vector into a matrix of dimension $N_x \times N_y$ and this is our recovered compressed image.

4. Computational Results

Unless specified we are working on an already compressed version of the image as specified in [4] and use gray scale. The size of the image is 53×41. This is the image in Fig. 1.

4.1. **Compressibility.** In order to exploit the sparsity that we discussed in Sec. 2.2 we look at the amplitudes of the DCT coefficients of $\hat{f}$. If we have few large coefficients that represents the ability to compress our image down to a limited amount of coefficients. The magnitude of coefficients of the forward DCT of our image vector have a mean of .1427 and standard deviation of .6107. We only have 14 coefficients that lie outside of two standard deviations from the mean. This signals to us that we should be able to compress the image by a large amount and still have a coherent structure/similar image. Fig. 2 is a histogram of the coefficient magnitudes of the DCT of our image vector. Note that there is a single coefficient lying at 26.7928 which is not shown in this graph. This indicates that our problem is well suited for compression. In Fig. 3 we plot the reconstructed images while only keeping the top 5, 10, 20, and 40 percent of DCT coefficients. We can see that for $P = 20$ we have a fairly good representation of our original image and $P = 40$ provides a decently clear compression while being able to throw away 60% of the data. Table 1 shows the number of coefficients that we keep for these top percentiles which displays the amount of sparsity in the Fourier basis. The hat and apple are fairly well represented even in the

Top P Percentile	# of Coefficients kept
5	109
10	218
20	435
40	869

TABLE 1

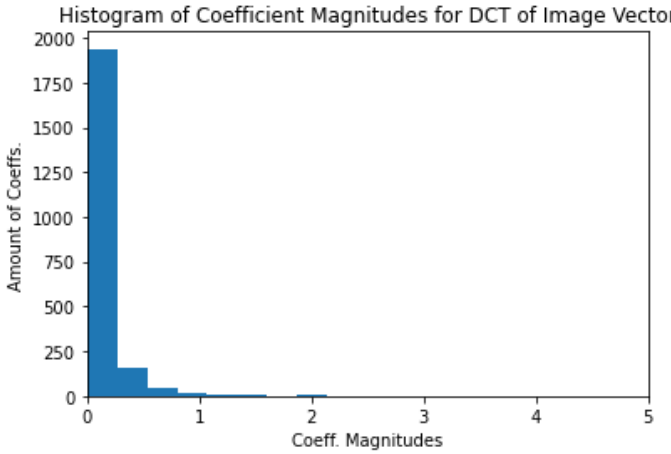

FIGURE 2. Histogram of coefficients from the DCT of our original image.

$P = 5$ case. The facial structure tends to blend with the background - this could be an artifact of the gray scale making the color of the face more similar to the background. This also helps to explain why the black hat, darker gray apple, and dark suit are well defined in the compression.

4.2. **Compressed Image Recovery.** We use several different amounts of measurements, M, which are a proportion, r, of the total size of the image vector N. In Fig. 4 we display three trials for each proportion r. We run three different trails so that we use three different randomly selected permutation matrices in case we get happen to choose a really good or bad permutation matrix. We see a fairly significant jump in clarity as we increase the number of measurements which should be expected. We can see some of the effects from choosing a different permutation such as the difference for $r = .4$ and trial 2 and 3. Similar to the compression problem even in the the $r = .2$ case we can recover the structure of the hat well. The apple and suit seem to be well recovered, however, the face tends to blur in with the backdrop and is difficult to recover. In Fig. 5, we use the same algorithm to recover a compressed image of Nyan Cat.

5. SUMMARY AND CONCLUSIONS

We explored the sparsity of our image in the DCT basis and showed that this was a good basis to perform compression on our image since there were very few large DCT coefficients. We compress by keeping the top percentile of coefficients. We also perform an image recovery of a compressed image by constructing several different a measurement matrix and solving a convex optimization to recover the corrupted/compressed image. We also used this algorithm to recover a mystery image from a given measurement matrix.

Future considerations we would like to consider are looking at different basis and exploring the sparsity of different problems in different bases.

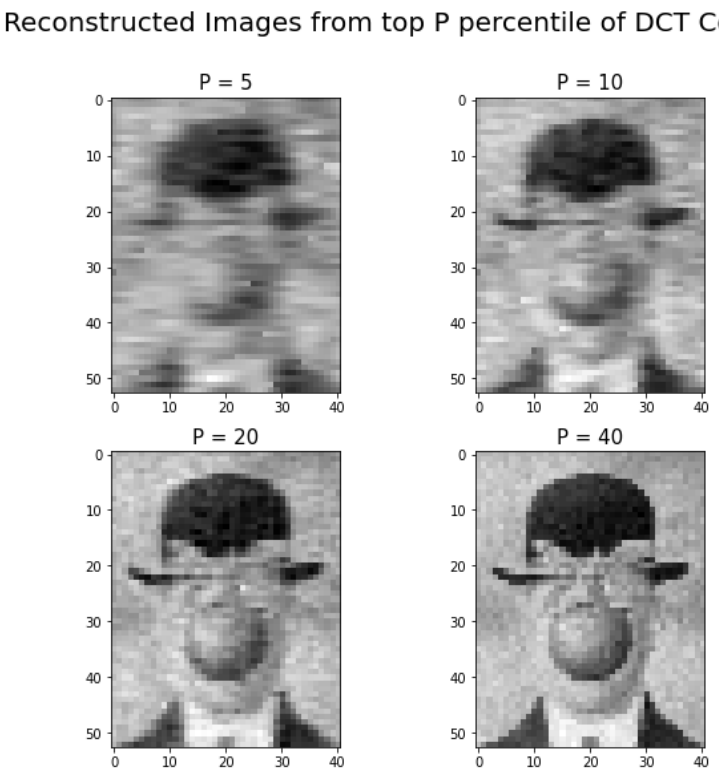

FIGURE 3. Reconstructed images using only the top P percentile of DCT coefficients.

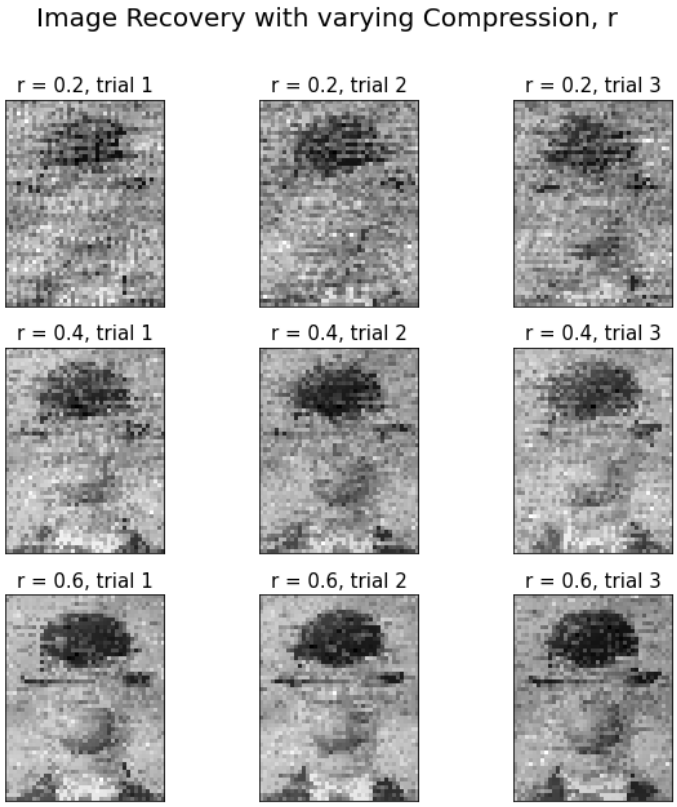

FIGURE 4. Reconstruction of compressed image with varying amount of measurements.

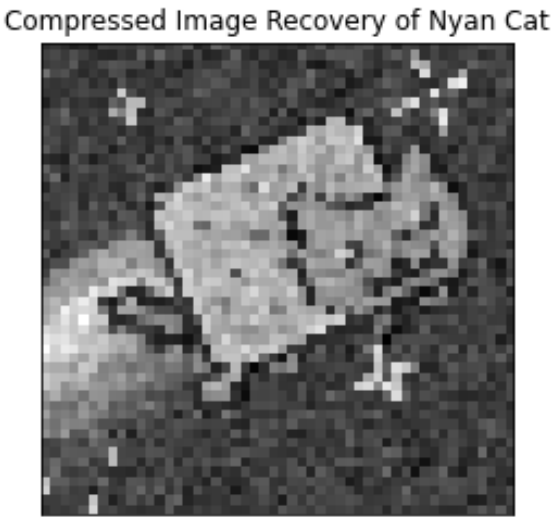

FIGURE 5. Reconstructed Nyan Cat from compression.

ACKNOWLEDGEMENTS

The author thanks fellow classmate Daniel Leon for discussions about the theoretical aspects of the sparse representation in the wavelet basis and for helpful discussions about the flow of the algorithm.

REFERENCES

[1] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
[2] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.
[3] B. Hosseini. Amath 582 assignment 5 helper code. Code to start assignment 5 of Amath 582.
[4] B. Hosseini. Amath 582 assignment 5 sheet. Sheet that contains relevant information for Assignment 5.
[5] B. Hosseini. Amath 582 course notes. Notes for Computational Methods for Data Analysis.
[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
[8] P. Raybaut. Spyder-documentation. *Available online at: spyder-ide.org*, 2009.
[9] J. Staggs. Amath 582 assignment 1. Homework assignment 2 for amath 582.
[10] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.
[11] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.