University of Colorado - Boulder

APPM 3310 Matrix Methods and Applications

July 18^{th} 2022

APPM 3310: Single Spectrum Analysis and Image Reconstruction

Author:
Courtney Hogan

Author:
Elizabeth Rodriguez

Professor:
Robert Benim

Author:
Steven Young

University of Colorado
Boulder

Contents

I. Abstract

At first glance, Single Spectrum Analysis appears to be a straightforward method of breaking down a large quantity of data and transforming it to a manageable set to be analyzed or interpreted. However, further investigation reveals it is not the single purpose tool it appears to be, but a multifaceted method with innumerable versatile applications. For the purpose of this paper, its function in regard to image processing is explored. Using core principles from linear algebra such as matrix manipulation, singular value decomposition, and calculating eigenvectors the team runs through a detailed example of how the process of singular spectrum analysis can be used to reconstruct an image that has been corrupted with noise. In doing this, two parameters are varied so their effect can be properly observed. These parameters include the window size used in constructing the trajectory matrix and the number of normalized eigenvectors used in the reconstruction. Following the implementation of this algorithm, the results do not exactly align with the proposed theory as increasing the number of eigenvectors used does not yield the expected outcome, which is the increase in accuracy of the reconstruction. However, decreasing the window size does behave as expected. Regardless, the team proves that single spectrum analysis lives up to its reputation as a powerful analytical tool.

II. Attribution

Steven designed the code, manipulated parameters to optimize efficiency, and helped analyze data. Elizabeth formulated the mathematical process and analyzed data produced by the code. Courtney did background research to introduce the topic and analyzed data produced by the code. Each individual contributed to all sections and discussed as a team to develop conclusions.

III. Introduction

Science and art are typically thought of as being polar opposites. While this statement holds truth in a traditional sense, there is more to gain from noticing their similarities. In recent years, considerable areas pertaining to both science and art have gone digital, utilizing technological strides in their respective fields. This notable commonality forms the basis for the fields to unite and advance. Photography, for example, has long been deemed to be an art rather than a science. Upon closer inspection, photographs rely heavily on principles of linear algebra and matrix methods. One particularly useful process photography has found tremendous value in is singular spectrum analysis.

Further investigation of the relationship between art and science illuminates the crucial role singular spectrum analysis (SSA) plays in digital photography. At its core, a photograph can be broken down into a two dimensional

matrix. Each entry holds a color value which corresponds to its respective pixel location. These values are consistently corrupted by blur and noise. It is borderline impossible to capture images without the interference of noise. Noise may take different forms,two of which are Gaussian noise and Salt & Pepper noise. While types and magnitudes of noise differ, the issue posed can be remedied through the implementation of single spectrum analysis. The effect of these imperfections can be observed through an algorithm, the basics of which are relatively simple.

To perform the algorithm it is useful to have a fundamental foundation of matrices: matrix multiplication, matrix transposes, eigenvalues, etc. This knowledge is applicable to singular value decomposition which plays an important role in SSA. It is essentially a factorization of a matrix in which eigenvalues and eigenvectors (which must be normalized) can be calculated and extracted to be used in other calculations. This information will be useful in the mathematical formulation as these normalized eigenvectors will act as parameters to be varied. Lastly, a critical tool will be the comparison of various squared Frobenius Distances. The Frobenius distance can be calculated using *Equation 5*. This equation will go on to calculate the distance between an image and the reconstruction from a corrupted version. This will be an extremely helpful tool in analyzing the effect of a given rendition of SSA as the squared Frobenious distance is directly related to similarity or dissimilarity between an original image and one that has been reconstructed.

Having set up the knowledge of concepts pivotal in understanding single spectrum analysis, an adequate knowledge of the SSA algorithm must follow. This requires a deep dive into the mathematical formulation that will form the backbone of the code and analysis. The final step towards understanding the importance of SSA is to apply the set of previously discussed information and go through the process using a digital image to examine the results firsthand. In this paper, a black and white image will be used for simplicity as to not distract from the analysis. The process of using single spectrum analysis will display its versatility and prove the advantage of using it to remedy corrupt images.

IV. Mathematical Formulation

This section will provide an in depth look into how Singular Spectrum Analysis can be used to manipulate images by either adding or removing noise. Images are represented by matrices. If the image is black and white, each entry of the matrix will be a level of grey corresponding to its location in the image. For a color image, each element contains a set of red, blue, and green levels that dictate the color at each respective location. These matrices take the place of time series which is typically used in traditional SSA.

Steps for Single Spectrum Analysis for Image Processing

First, once the image is loaded into MATLAB, the image's data can be stored in the two dimensional h x w matrix $\mathbf{I}$, the image matrix. However, there is not much use for the image matrix as is so it must be transformed into a trajectory matrix, denoted $\mathbf{X}$. This proves more useful in future calculations as it contains the data provided in the image matrix in addition to the relationship between pixel locations.

To construct the trajectory matrix from the image matrix the notion of a "window" must be implemented. The window has size u x v, where:

$$1 \leq u \leq h$$

$$1 \leq v \leq w.$$

It roves over the entire image matrix, gathering data from each element of $\mathbf{I}$. The reference point, (i,j), dictates where the window lies as it moves over $\mathbf{I}$. The range of all reference points can be described as:

$$1 \leq i \leq u = h - u + 1$$

$$1 \leq j \leq v = w - v + 1.$$

As the window moves around the image, the reference point moves accordingly. Its movement can be described as such:

$$(i, j) = \begin{cases} (i, j+1) & \text{if} j \leq w - v \\ (i+1, 1) & if\, j = w - v + 1, i \leq h - u \end{cases}$$

The window can now roam over $||I_{i+k-1,\, j+l-1}||_{k=1,\ldots,u, l=1,\ldots,v}$.

Each row of each window is transposed to form column vectors.

$$\vec{W} = (\vec{W_1^T}, ..., \vec{W_u^T})$$

Each of these column vectors are the windows, and are then transformed into

$$\vec{W_{i,j}} = (I_{i,j}, I_{i,j+1}, I_{i,j+2}, ..., I_{i,j+v-1}, I_{i+1,j}, I_{i+2,j}, ..., I_{i+1,j+v-1}, ..., I_{i+u-1,j+v-1}) \in \mathbb{R}^{\approx\lessapprox}$$

These new columns form the trajectory matrix **X** which is given by *Equation 1* below,

$$X = (W_{1,1}, W_{1,2}, ..., W_{1,v}, W_{2,1}, ..., W_{2,v}, ..., W_{u,v}) \tag{1}$$

and has size uv x $(h - u + 1)(w - v + 1)$.

Example set up of trajectory matrix:

Let the image matrix be hxw, and take a 2x2 window.

$$\begin{bmatrix} |I_{1,1} & I_{1,2}| & ... & I_{1,w} \\ |I_{2,1} & I_{2,2}| & ... & I_{2,w} \\ ... & ... & ... & ... \\ I_{h,1} & I_{2,h} & ... & I_{h,w} \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} I_{1,1} & ... & |I_{1,w-1} & I_{1,w}| \\ I_{2,1} & ... & |I_{2,w-1} & I_{2,w}| \\ ... & ... & ... & ... \\ I_{h,1} & ... & I_{h,w-1} & I_{h,w} \end{bmatrix} \rightarrow$$

$$\rightarrow ... \rightarrow$$

$$\begin{bmatrix} I_{1,1} & ... & I_{1,w-1} & I_{1,w} \\ ... & ... & ... & ... \\ I_{h-1,1} & ... & |I_{h-1,w-1} & I_{h-1,w}| \\ I_{h,1} & ... & |I_{h,w-1} & I_{h,w}| \end{bmatrix}$$

Therefore, $\vec{X}$ is

$$\begin{bmatrix} I_{1,1} & I_{1,2} & ... & I_{1,w} \\ I_{2,1} & I_{2,2} & ... & I_{2,w} \\ ... & ... & ... & ... \\ I_{h,1} & I_{h,3} & ... & I_{h,w} \end{bmatrix}$$

This matrix $\vec{X}$ is otherwise known as a Hankel matrix. A Hankel matrix can be defined as a square matrix with a constant skew-diagonal entry. The i, j entry of a Hankel matrix H is defined as

$$H_{i,j} = H_{i+k,j-k} \text{ for all } k = 0, ..., j - 1$$

$$\text{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_2 & h_3 & h_4 \\ h_3 & h_4 & h_5 \end{bmatrix}$$

Once the trajectory matrix ($\vec{X}$) has been calculated the setup for SSA is essentially done and singular value decomposition (SVD) can begin. The purpose of SVD is to calculate eigenvalues and their corresponding normalized eigenvectors which will be manipulated in later in this report. Recall the size of **X** is p x q, where p=uv and q = $(h - u+1)(w - v+1)$. In the process of SVD it is favorable to calculate product X^TX or XX^T that yields the matrix of smallest dimension. Note that in this case $p \leq q$ so naturally it is wise to compute the eigenvalues of product XX^T which yields a matrix of size pxp. From here the eigenvalues and their corresponding eigenvectors can be calculated. The

eigenvalues are given by λ_i where $i = 1, ..., p$. Dividing each eigenvector by its Euclidean norm yields the normalized eigenvectors which can be denoted by P_i. The decomposition of XX^T is thereafter given by *Equation 2*.

$$XX^T = \Sigma_{i=1}^{p} \lambda_i P_i P_i^T \tag{2}$$

The eigenvalues of XX^T, λ, are always greater than or equal to zero with P_i being its corresponding normalized eigenvector. It is important to recall here that the goal is to reconstruct the chosen image using a handful of these eigenvectors. Although all eigenvectors have been computed, it is not necessary to use all of them due to the nature of the process at hand. So, here a selection process begins where the number of eigenvectors to be used in reconstruction can be called *l*. Let *l* be an integer such that $1 \leq l \leq$ p. In theory, the more eigenvectors used in the de-noising process, the clearer the image will be. The image can now be reconstructed using *Equation 3*.

$$\bar{X} = \sum_{k=1}^{l} P_{ik} P_{ik}^T X \tag{3}$$

This yields a matrix that approximates **X** and facilities a new image matrix of the original size, *h* x *w*, by averaging the original image and corresponding entries of $\bar{X}$.

For an image matrix, we can measure the distance between the original image and the reconstruction of corrupted images (corrupted with noise). The most common additive stochastic noise would be Gaussian white noise, which is completely random.

$$Z_i \; N(0, \sigma^2) \tag{4}$$

To reconstruct the original image, again we use singular spectrum analysis with a different amount of eigenvectors. Measuring the distance between the original image matrix and the reconstructed matrix of the noisy image will dictate how close the noisy image is to the original, or in other words the dissimilarity of the two images. The most fitting method used to calculate the distance is the normalized squared Frobenius distance which is defined below.

$$d(I^{(1)}, I^{(2)}) = \left[\frac{1}{hw} \sum_{i,j=1}^{h,w} \left| I_{i,j}^{(1)} - I_{i,j}^{(2)} \right|^2 \right] \tag{5}$$

The smaller the distance the more similar the two images are to each other. To check the SSA process, it is good to note that the original image should have a smaller distance between the reconstructed images than between the noisy image. We can also note that

$$\lim_{d \to uv} d(I^{(1)}, I^{(2)}) = 0 \tag{6}$$

between the original image and the reconstructions. We can also measure the distance using singular value decomposition, that will increase the accuracy of the dissimilarity. First, let $X^{(1)}$ and $X^{(2)}$ be the pxq trajectory matrices for $I^{(1)}$ and $I^{(2)}$, respectively. Next we normalize $X^{(1)}$ and $X^{(2)}$ to get

$$Y_1 = X^{(1)} / \sqrt{tr(X^{(1)}(X^{(1)})^T)}, Y_2 = X^{(2)} / \sqrt{tr(X^{(2)}(X^{(2)})^T} \tag{7}$$

Now we are able to use singular value decomposition to obtain $Y_1 Y_1^T$ and $Y_2 Y_2^T$. From this, we are able to determine that these two products are positive-definite, therefore $\lambda_i \geq 0$ and $\mu_i \geq 0$ (the corresponding eigenvalues) for all i. We also know the sum of both λ_i and μ_i are equal to 1. We then create a normalized trajectory matrix $Y = (Y_1, Y_2)^T$ then apply SVD to obtain

$$YY^T = (Y_1, Y_2)^T (Y_1, Y_2) \tag{8}$$

We will label the eigenvalues of YY^T as $v_1 \geq v_2 ... \geq v_{(2p)} \geq 0$. Then, for any positive integer k it cant be said that:

$$\sum_{j=1}^{k} \lambda_j + \sum_{j=1}^{k} \mu_j \geq \sum_{j=1}^{k} v_j \tag{9}$$

We can use a cumulative distribution function $F(t)$ to define the sum of λ_j and the sum of μ_j as

$$F_1(t) = \sum_{j=1}^{|t|} \lambda_j, F_2(t) = \sum_{j=1}^{|t|} \mu_j \tag{10}$$

and therefore, can represent $F(t)$ as $F(t) = 1/2 \sum_{j=1}^{|t|} v_j$. From equation (9), we get that $F_2(t)$ - $2F(t) \geq 0$ for all t $\geq$ 0. From this, we get

$$G(t) = F_1(t) + F_2(t) - 2F(t) \tag{11}$$

which tells us how similar $I^{(1)}$ and $I^{(2)}$ are to each other. If G(t) is large than we know that $I^{(1)}$ is different from $I^{(2)}$, similarly if G(t) is small we can conclude that the images are similar to each other. We can now define the distance as

$$d(I^{(1)}, I^{(2)}) = \int_0^k G(t)dt = \sum_{j=1}^{k} (\lambda_j + \mu_j - v_j) \tag{12}$$

Following this algorithm provides everything needed to add or remove varying amounts of noise from a photo. The next steps in this process are to manipulate the eigenvectors and use those to reconstruct the image in question, the process of which will be expanded upon in the next section.

V. Examples and Numerical Results

The clearest way to demonstrate how single spectrum analysis truly works in deconstructing a reconstructing an image is through an example. A code was formulated in MATLAB (see Appendix A) that closely mirrors the process outlined in section **IV**. The test subject will be a square image of a chicken (see Figure 3). This image is randomly blurred using Gaussian noise and Salt & Pepper noise at varying intensities of 0.001, 0.01, 0.05, and 0.15. From here, the experiment seeks to explore the effect on image reconstruction from varying two critical parameters used in the mathematical formulation: The window size and number of eigenvectors used. In theory a smaller window size and larger amount of eigenvectors used will yield the more accurate reconstruction. Although the distinction between the original and corrupted photo is apparent to the naked eye, visual comparison is riddled with random and systematic error. To eliminate these sources of error from the analysis, similarity between images will be measured using the squared Frobenious distance between original and reconstructed.

Table 1 below shows the squared Frobenious distances as a result of the algorithm run using a window of size *10*x*10*. This window size is used in the calculation of both Gaussian and salt & pepper noise at varying degrees of intensity and using different numbers of eigenvalues. Based on a theoretical knowledge of the process, the smallest Frobenious distance between reconstruction and original should occur when the number of eigenvectors used is at its maximum, 10. It is clear in the table that this is not precisely the case. In each example of reconstruction, the most accurate reconstruction occurs at varying numbers of eigenvectors. In all cases the reconstruction of the noised image is more accurate to the original image than the noised image is itself, except for the two cases of most minimal noise applied, Gaussian 0.001 and Salt & Pepper 0.001.

Table 1

Number of Eigenvectors Used (l)	Original Image	Gaussian (0.001)	Gaussian (0.01)	Gaussian (0.15)	Salt & Pepper (0.001)	Salt & Pepper (0.05)	Salt & Pepper (0.15)
0	0	60.20	664.91	2006.25	33.66	980.40	2740.19
1	610.48	614.13	627.21	1042.98	607.52	607.72	604.25
2	611.35	615.00	628.18	1043.8	608.38	608.56	605.05
3	611.35	615.00	628.17	1043.81	608.38	608.57	605.05
4	611.13	614.78	627.89	1043.56	608.16	608.36	604.88
5	611.14	614.79	627.91	1043.56	608.18	608.38	604.89
6	611.12	614.77	627.87	1043.54	608.15	608.35	604.87
7	611.14	614.78	627.89	1043.54	608.17	608.37	604.89
8	611.14	614.78	627.89	1043.54	608.17	608.37	604.89
9	611.14	614.78	627.89	1043.54	608.17	608.37	604.89
10	611.13	614.78	627.88	1043.54	608.17	608.37	604.88

Table 1 Frobenius Distance between original image and reconstructed noised image using 10x10 window

Table 2 below uses the same parameters as Table one with the only difference being the window size. The size here has been decreased from *10*x*10* to *4*x*4*. The expected result from this manipulation is a decrease in Frobenious distance between reconstructed and original distance compared to the values in Table 1. Also the original corrupted images display the same increase in distance as the intensity of corruption increases as is shown in Table 1. Although the setup is correct, the same issue with inconsistent results regarding distances and changing eigenvectors presents itself.

Table 2

Number of Eigenvectors Used (l)	Original Image	Gaussian (0.001)	Gaussian (0.01)	Gaussian (0.15)	Salt & Pepper (0.001)	Salt & Pepper (0.05)	Salt & Pepper (0.15)
0	0	61.59	680.81	2059.72	34.20	1004.54	2816.10
1	247.81	250.93	258.46	964.01	246.80	301.77	451.33
2	248.01	251.13	258.66	964.17	247.00	301.95	451.49
3	248.03	251.15	258.67	964.15	247.02	301.97	451.50
4	247.97	251.08	258.62	964.14	246.96	301.92	451.46
5	247.97	251.09	258.62	964.16	246.96	301.92	451.46
6	247.96	251.08	258.62	964.16	246.96	301.92	451.47
7	247.97	251.09	258.62	964.17	246.96	301.92	451.47
8	247.97	251.09	258.62	964.17	246.96	301.92	451.47
9	247.97	251.09	258.62	964.18	246.96	301.92	451.47
10	247.97	251.09	258.62	964.18	246.96	301.92	451.47

Table 2 Frobenius Distance between original image and reconstructed noised image using 4x4 window

Figure 1 is shown below. The purpose of Figure 1 is to illustrate the effect reconstruction has on a corrupted image. The data used corresponds to columns 2 and 4 in Table 1. The orange line represents a constant: the distance between the original image and the corrupted image before its reconstructions. The blue line shows the distance between the original and the noised image as it is reconstructed using an increasing number of eigenvectors. The gray line shows the

distance between the original image and its own reconstructions using an increased number of eigenvectors. Comparing the blue and grey lines, it is clear that the increase or decrease with each eigenvector are similar in the way that there is no sizeable change in value as the number of eigenvectors increase. In other words, the average distance between the blue line and orange line is constant as the eigenvectors change. The same can be said about the grey and orange lines.

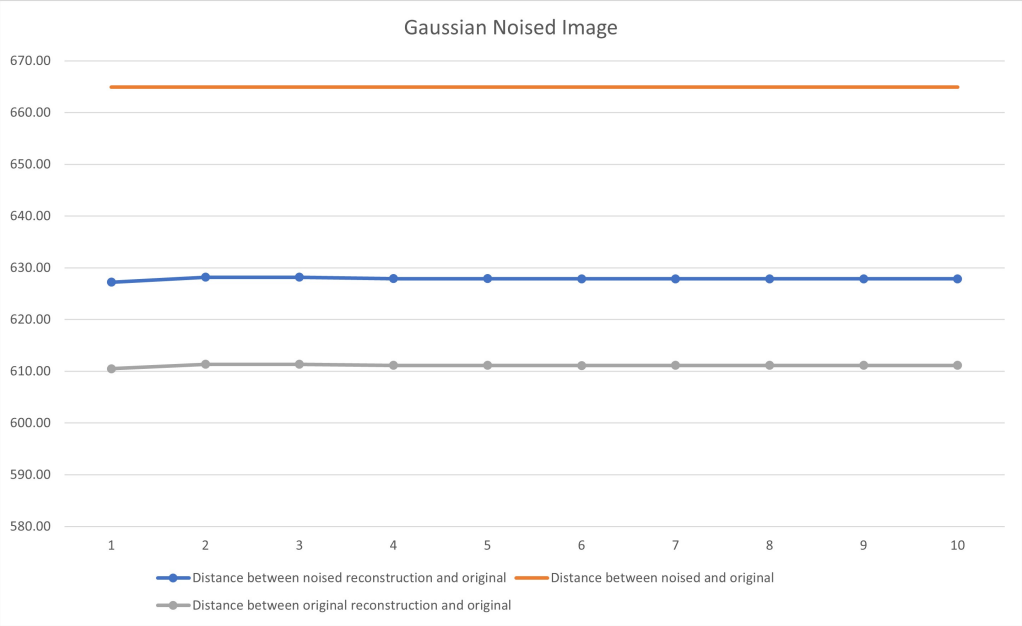

Fig. 1 Gaussian Reconstruction distances

Similarly, Figure 2 uses data from Table 1 to exemplify how reconstructions of the Salt & Pepper image with noise intensity 0.05 compares to reconstructions of the original image, corresponding to columns 2 and 7 of Table 1. Color-wise everything is the same here except the blue line now represents the reconstructions of the Salt & Pepper noise as opposed to the Gaussian noise. It is important to note here that although the same trend represents itself here of no increased accuracy as the number of eigenvectors is increased, the reconstructions of the noised image are more accurate to the original than the reconstructions of the original itself.

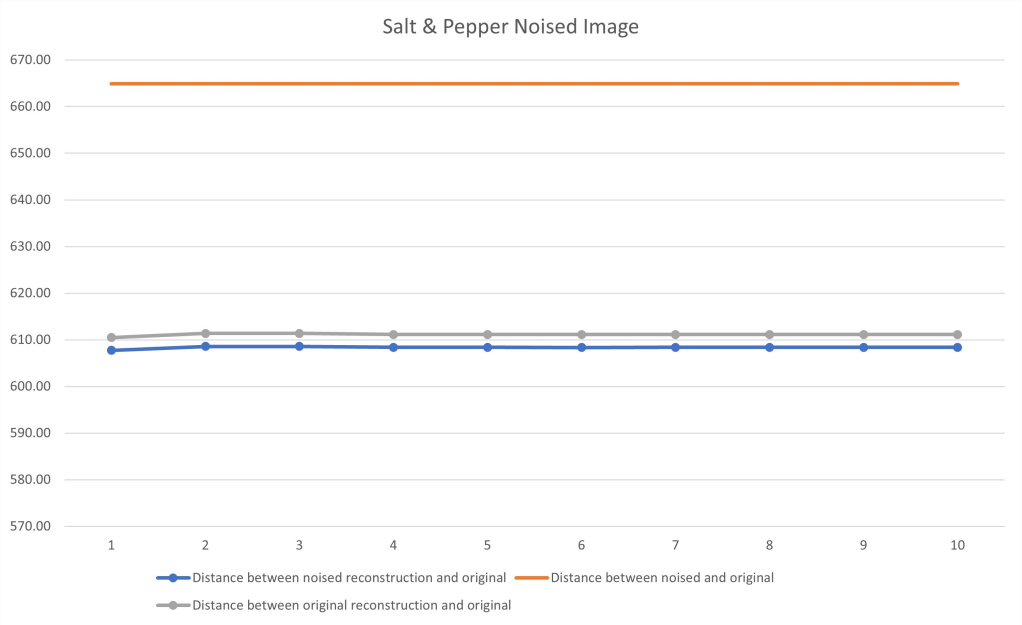

Fig. 2 Salt & Pepper Reconstruction distances

Fig. 3 Original image used for experiment

Figure 3 shows the original image used in all of the experimentation conducted. Below in Figures 4 and 5, the different intensities of both Gaussian noise and Salt & Pepper noise can be seen respectively. From left to right, the intensities range from the least amount of noise to the greatest amount of noise.

Fig. 4 Original image blurred with Gaussian noise at different intensities

Fig. 5 Original image blurred with Salt & Pepper noise at different intensities

Figures 6, 7, and 8 each show the image used in the experimentation, and the resulting respective reconstruction using 1, 5, and 10 eigenvectors. It can be seen both visually and by the data in Table 1 that there is virtually no difference between using 1 eigenvector and 10 eigenvectors in the reconstruction of the image. The expected result is that as the number of eigenvectors used in the reconstruction is increased, the reconstruction becomes more accurate to the original, non-noised image, and even surpasses the accuracy of the noised image itself within 10 eigenvectors. This is not the case here, however it should be noted that the reconstructions in most cases are more accurate to the original image than the noised image is itself from the start, resulting in a net increase in accuracy of the reconstruction.

Fig. 6 Original image and reconstructions of original using 1, 5, and 10 eigenvectors

Fig. 7 Gaussian noised image and reconstructions using 1, 5, and 10 eigenvectors

Fig. 8 Salt & Pepper noised image and reconstructions using 1, 5, and 10 eigenvectors

VI. Discussion and Conclusions

The purpose of this paper was to prove the effectiveness of single spectrum analysis as a tool for reconstructing corrupted images. While the process and application are explained in detail, putting the algorithm into practice did not yield the expected results.

Based on a theoretical knowledge of the process, the smallest Frobenious distance between reconstruction and original should occur when the number of eigenvectors used is at its maximum, 10. It is clear in both Table 1 and Table 2 that this is not precisely the case. Looking at Table 1, in each example of reconstruction (with the exception of intensity level 0.001), the distance drops significantly between the original corrupted image and its reconstruction using one eigenvector, but then increases or decreases slightly with each following reconstruction using more eigenvectors. The net effect between reconstructions using 1 eigenvector and 10 eigenvectors should be a drop in distance. However, the observed effect is a distance gain. The same is true for images of corruption intensity 0.001, except the jump from the original corrupted image to its reconstruction using one eigenvector is a considerable increase. Additionally, the most accurate reconstruction of the image takes place at varying numbers of eigenvectors, which does not follow with the proposed theory. This inconsistency is independent of the window size as it appears in both Tables 1 and 2.

There are a few potential sources for this error other than a bug in the code. To assess what has gone wrong in the code, the factors that did behave properly must be examined. Starting at the beginning, it is clear that the corrupted images are set up correctly. It follows in logic and theory that images that have been more intensely corrupted will have a larger Frobenious distance than those that have been only slightly corrupted. Looking at the images corrupted with Gaussian noise in the first row of Table 1, it is clear that the distance increases with intensity as it increases from 0.001 to 0.01 to 0.15 and 0.001 to 0.05 to 0.15.

Recall, the main distinction between the data in Table 1 and Table 2 is the size of the window used. Earlier, it was stated that a smaller window size would yield a more accurate reconstruction. Table 1 used window size *10*x*10* while Table 2 ran the algorithm with a window size of *4*x*4*, so theoretically the values in Table 2 should be smaller than their corresponding values in Table 1. Comparing the two, column by column shows Table 2's initial distances are slightly larger than Table 1 but average values of each column are distinctively lower than those of Table 1. This relation allows two conclusions to be drawn. First, the team can conclude that using a smaller window size does in fact create a more

accurately reconstructed image. Essentially, this section of the theory has been proved. Secondly, the section of the code pertaining to the windows is functioning properly and not causing the error.

The central components of the experiment can be broken down into the following parts: creating the trajectory matrix $\mathbf{X}$ using a window, using $\mathbf{X}$ to calculate XX^T and perform singular value decomposition, using the normalized eigenvectors to construct the matrix $\bar{X}$, which is an approximation for $\mathbf{X}$, then use $\bar{X}$ to reconstruct the corrupted images, and calculating the squared Frobenious distances between the original and reconstructed images. Looking at the process like this, it should be relatively easy to find the error. The trajectory matrix, singular value decomposition, the matrix , and the Frobenious distances all appear to be functioning properly so the most logical conclusion to draw is that something has gone somewhat awry with the reconstruction step itself. Unfortunately, the team was not able to find a solution to the code but spent extensive time troubleshooting and performing error analysis testing.

Taking a step back from the surplus of information that precedes this, we are again faced with the initial proposed question: What is the effectiveness of singular spectrum analysis in image processing? It was hypothesized from the start that SSA is an extremely powerful tool in situations such as this one where an image is corrupted with noise and must be reconstructed. Background research informed this hypothesis and gave the team direction as to how best test this theory. Although the team's experiment did not yield results that directly supported their theory, extensive error analysis indicates that solving the bug in the code would produce results consistent with background research. Accordingly, it can be concluded with relative certainly that SSA is, in fact, a practical tool for image reconstruction.

VII. References

Rodriguez-Aragon, Licesio J, and Anatoly Zhigljavsky. “Singular Spectrum Analysis for Image Processing.” Statistics and Its Interface , vol. 3, International Press, Boston, MA, 2010, pp. 419–426.

Zhigljavsky, Anatoyl. “Singular Spectrum Analysis for Time Series: Introduction to This Special Issue.” Statistics and Its Interface, vol. 3, International Press, Boston, MA, 2010, pp. 255–258.

VIII. Appendix

A. Matlab Code

```
%% APPM 3310 Project
% Steven Young
% Courtney Hogan
% Elizabeth Rodriguez
% 18July2022


%% Housekeeping
clear all;
close all;
clc;


%%
% Reading in image into matrix form and separating the colors

% Reading in original, non-noised image
% filename = 'square.jpg';
% [original, reds, greens, blues] = colors(filename,1);
original = rescale(im2double(imread('Square_gray.jpg')),0,255);
figure(1)
imshow(uint8(original))
colormap('gray')


% Code to apply noise to the image
% grays = rescale(original,0,1);
% grays = imnoise(grays,'gaussian');
% imwrite(grays,'Square_blur_1.jpg')

% Reading in noised image from file
noised = im2double(imread('Square_blur_1.jpg'));
noised = rescale(noised,0,255);
figure(2)
imshow(uint8(noised))
colormap('gray')


%% Constructing the trajectory matrix
% Height and width of original image
h = size(original,1);
w = size(original,2);

% Window size used
u = 10;
v = 10;
```

```
47
48  % Iterating through the noised image, building the trajectory matrix
49  % Moving the window left to right, top to bottom
50  idx = 0;
51  for i = 1:h-u+1
52     for j = 1:w-v+1
53         W = noised(i:i+(u-1),j:j+(v-1));
54         for x = 1:u
55             C(x+idx:(x+idx)+(u-1),1) = W(x,:)';
56             idx = idx+(u-1);
57         end
58         idx = 0;
59         % W_bar is all elements of a window, written in a single column
60         % vector
61         if j==1
62            W_bar = C;
63         else
64            W_bar = [W_bar C];
65         end
66     end
67     % X is trajectory matrix
68     if i==1
69         X = W_bar;
70     else
71         X = [X W_bar];
72     end
73  end
74
75
76  %% SVD
77  % Height and width of trajectory matrix, p x q, corresponding to paper
78  p = u*v;
79  q = (h-u+1)*(w-v+1);
80
81  % Matrix for applying SVD
82  A = X*X';
83
84  % Applying SVD, P and V are left and right normalized eigenvectors, S is
85  % the eigenvalues
86  [P,S,V] = svd(A);
87
88
89  %% Eigenvectors
90  % Number of eigenvectors used to reconstruct
91  L = 1;
92  % Computing X_bar, using L eigenvectors, which is an approximation for X
93  for i = 1:L
94      if i==1
95          X_bar = P(:,i)*P(:,i)'*X;
96      else
97          temp = P(:,i)*P(:,i)'*X;
98          X_bar = X_bar +temp;
99      end
100 end
101 X_bar = rescale(X_bar,0,255);
102
103
104 %% Reconstruction
105
```

```
% Averaging over the corresponding entries of X_bar
RC = zeros(1,q);
for i = 1:u*v
   RC = RC + X_bar(i,:);
end
RC = RC/(u*v);

% Reshaping RC into the right size for the image
% Image reconstructed from noised image
I_rc = reshape(RC,q/(h-u+1),q/(w-v+1))';
% Scaling to ensure values of the reconstructed matrix are [0 255]
I_rc = rescale(I_rc,0,255);


imwrite(rescale(I_rc,0,1),'Square_sp_10eig.jpg')
% Displaing image reconstructed from noised image
figure(3)
imshow(uint8(I_rc))
colormap('gray')


% Calculating Frobenius distances to see efficiency of SVD reconstruction
D = 0;
D_o = 0;
for i = 1:h-u+1
    for j = 1:w-v+1
      % Distance between original image and reconstructed noised image
      D = D + (original(i,j) - I_rc(i,j))^2;
      % Distance between original image and noised image
      D_o = D_o + (original(i,j) - noised(i,j))^2;
    end
end
D = D/(h*w);
D_o = D_o/(h*w);


%% Functions
% Function to separate given image into gray and the 3 colors
function [Gray, red, green, blue] = colors(filename,Z)

    img = imread(filename);
    red = img(:,:,1);
    green = img(:,:,2);
    blue = img(:,:,3);

    a = zeros(size(img, 1), size(img, 2));
    Red = cat(3, red, a, a);
    Green = cat(3, a, green, a);
    Blue = cat(3, a, a, blue);

    img = double(img);
    red = img(:,:,1);
    green = img(:,:,2);
    blue = img(:,:,3);

    Gray = img(:,:,1)/3 + img(:,:,2)/3 + img(:,:,3)/3;

    figure(Z)
    subplot(2,2,1)
```

```
    imshow(uint8(Gray))
    colormap('gray')
    subplot(2,2,2)
    imshow(Red)
    subplot(2,2,3)
    imshow(Green)
    subplot(2,2,4)
    imshow(Blue)

end
```