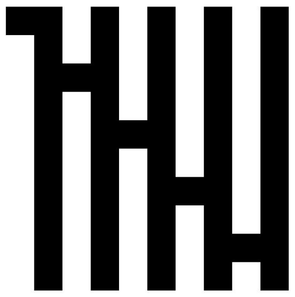
Oscar Alvarez and Hannah Grant
Texas Woman's University
June 2016

Applying the SVD to Image Compression

A digital image processor's ability to construct and alter images is grounded in the coding methods of computing with linear algebra. The primary methods of linear algebra used in image processing involve the manipulation and application of matrices. In this section, we will take a black and white image and use the strategy of finding the SVD (singular value decomposition) to achieve image compression by lowering the quality of the image meanwhile reducing the memory required for storage. Using these methods will help us find a balance that will minimize the amount of distortion while reducing memory space.

Here is an image created using MATLAB by applying some of the concepts of linear algebra.



This image was constructed using a matrix by defining its values to either the color white or black. This is just a simple example of the many possibilities of image manipulation with MATLAB.

1 Defining a matrix

A matrix is a useful tool when storing and managing large systems of data. A matrix is represented as $A_{(ij)}$, where i is the number of rows and j is the number of columns. The values inside a matrix are defined as being the elements of a matrix, or a_{ij}.

$$A_{(ij)} = \begin{bmatrix} a_{11} & a_{12} & . & . & . & a_{1j} \\ a_{21} & & & & & \\ . & & . & & & \\ . & & & . & & \\ . & & & & . & \\ & & & & & \\ a_{i1} & & & & & a_{ij} \end{bmatrix}$$

This example matrix below is represented as $C_{(3x2)}$,

$$C = \begin{bmatrix} 2 & -1 \\ 12 & 26 \\ 4 & 9 \end{bmatrix}$$

1.1 Matrix rank

Another term we need is the rank of a matrix. We can say the rank of the matrix is the number of nonzero rows once it has been reduced into row echelon form. Since our image matrices will always be reduced, we can say that the rank of our matrices will be the same as the number of rows of our matrix. Given the example matrix R,

$$R = \begin{bmatrix} 2 & -1 \\ 12 & 26 \end{bmatrix}$$

the rank of this matrix is 2.

2 The Singular Value Decomposition (SVD)

$$A = U\Sigma V^T$$

The SVD is defined as the singular value decomposition of a matrix. This means we can decompose a matrix into three smaller matrices. In the process, we can extract and create a diagonalized matrix which will contain our singular values. This singular value matrix will be represented by Σ in the SVD equation shown above.

We need a matrix to decompose. It must be able to be diagonalized and must not have repeating eigenvalues. We represent it as A. We use A to create the three matrices in the equation.

$$A = \begin{bmatrix} u_{11} & u_{12} & . & . & . & u_{1k} \\ u_{21} & . & & & & \\ . & & . & & & \\ . & & & . & & \\ . & & & & & \\ u_{i1} & & & & & u_{ik} \end{bmatrix} \begin{bmatrix} s_{11} & 0 & . & . & . & 0 \\ 0 & s_{22} & & & & \\ . & & . & & & \\ . & & & . & & \\ . & & & & & \\ 0 & & & & & 0 \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & . & . & . & v_{1k} \\ v_{21} & . & & & & \\ . & & . & & & \\ . & & & . & & \\ . & & & & & \\ v_{i1} & & & & & v_{ik} \end{bmatrix}^T$$

- The matrix U is formed by the eigenvectors of AA^T.
- Σ is the matrix with the singular values of A on the diagonal.
- Matrix V is formed by the normalized eigenvectors of A^TA.

3 MATLAB

MATLAB software is a powerful tool mathematicians can use when working with large matrices. We will be using MATLAB to alter an image and go over some basic coding structures.

3.1 Basic Coding Structures

In this section we introduce the main structure we need to manipulate an image and then, using MATLAB, show how to import and alter an image.

3.1.1 The "For" Loop

A "for" loop is a command in MATLAB used to repeat a block of code for certain values or a given amount of times. In a sense, we are constructing a row vector to determine how many times and what values we want to loop the block of code.

For example, when we type

```
k=1:6
```

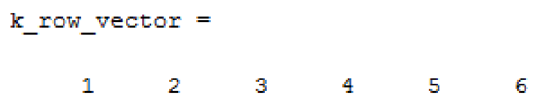
the result is:

```
k =

     1     2     3     4     5     6
```

We are actually creating a row vector that has the values of 1 through 6. If we wanted to create a row vector with specific values, we can create one by inserting the numbers in brackets only separated by a comma.

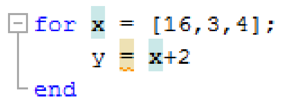
```
k_row_vector = [1,2,3,4,5,6]
```

Result:



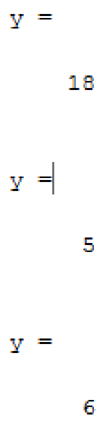
```
k_row_vector =

     1     2     3     4     5     6
```

Let us say that we want to solve the function $y = x + 2$ for the numbers 16, 3, and 4. We can use a for loop to create a row vector with these values. In the block of code inside of the “for” loop, we would insert our equation that we would want to solve for. This will run the equation for the specified amount of times and for the specific values. Here is what a “for” loop would look like in this situation.



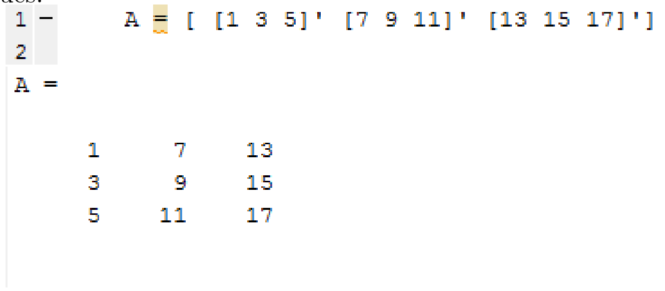
```
for x = [16,3,4];
    y = x+2
end
```

Our results will end up to be the values as requested.



```
y =

    18


y =

     5


y =

     6
```

Now we will do an example of a for loop using a matrix. First we will construct our matrix A. Then we will use a “for” loop to change some of our values.



```
      A = [ [1 3 5]' [7 9 11]' [13 15 17]']

A =

     1     7    13
     3     9    15
     5    11    17
```

Let us say that we want to change the values of columns 2 through 3. We create a “for” loop for our matrix that only changes columns 2 through 3 using an equation.

```
for j=2:3,
    A(j,:) = j^2+3
end
```

These are our results,

```
A =

     1     7    13
     7     7     7
     5    11    17


A =

     1     7    13
     7     7     7
    12    12    12
```

We can also place loops inside of loops which will make more sense once we begin applying the “for” loop in our SVD project code.

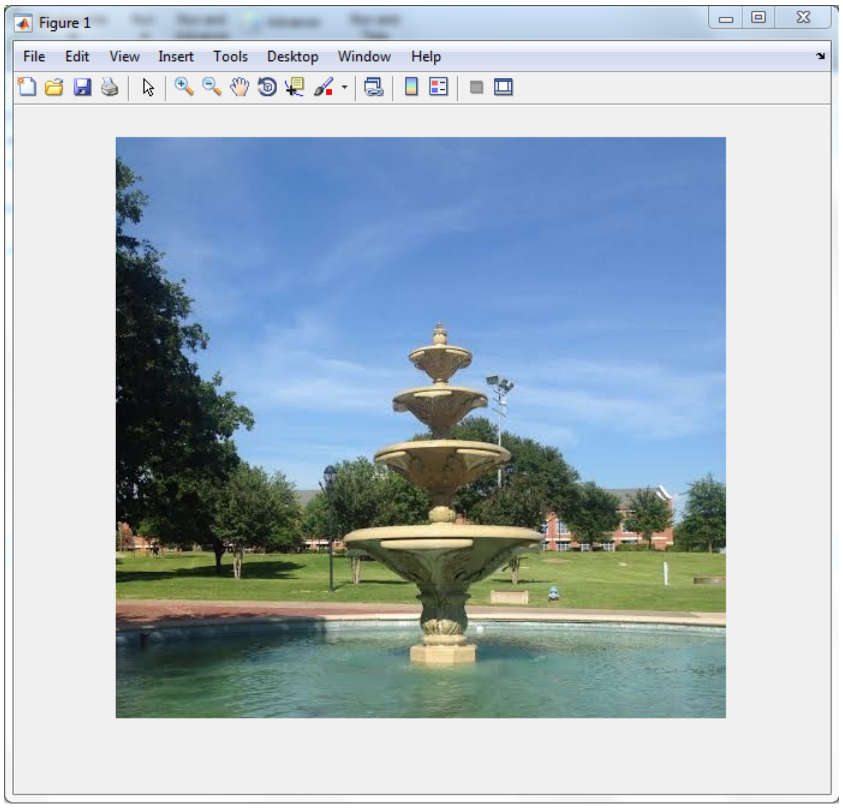
3.1.2 Importing/Altering an Image

First, we begin by saving the image in a location where MATLAB can find it. Usually, it would be a good practice to create a folder where you will be keeping all of your MATLAB files for the particular project. To import an image into MATLAB, we will need to run the following command:

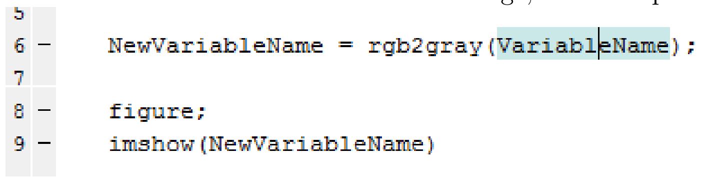
```
VariableName = imread('fountain.jpg');
```

This will import our image and save it as a new variable that we can use. If we want to view our image, you can display it in a new window where it will be represented by a figure.

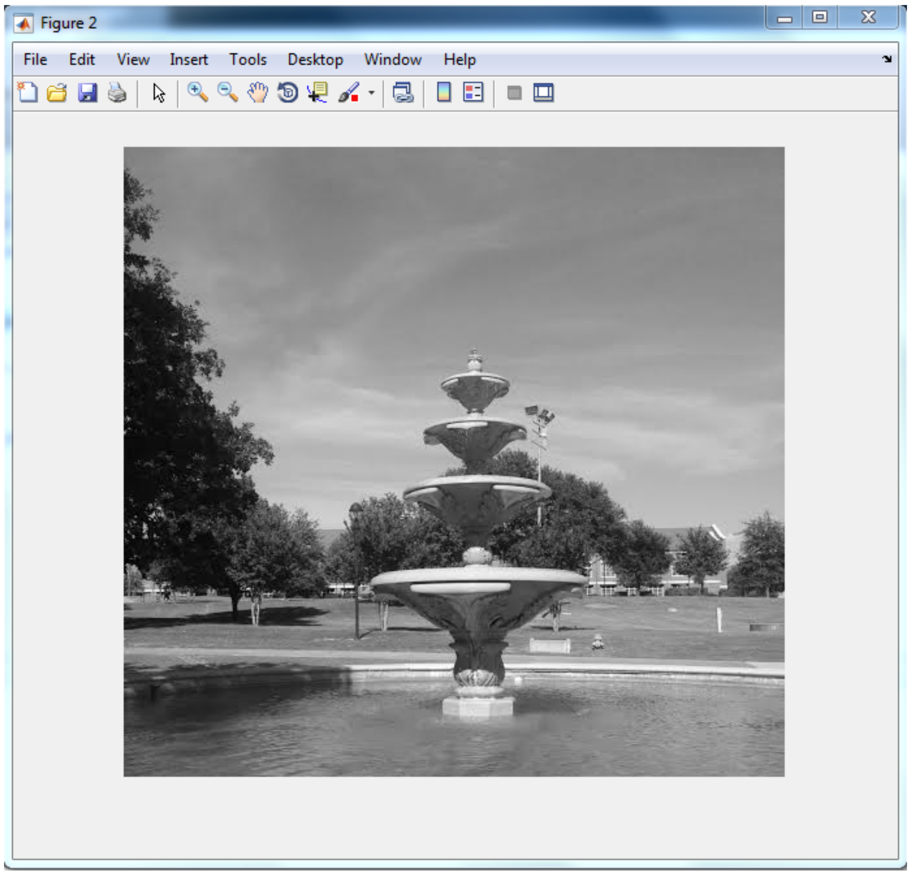
```
figure;
imshow(VariableName)

```



We can now alter the image with code directly. Here is an example; in order to create a black and white version of the image, we can input the command



```
NewVariableName = rgb2gray(VariableName);

figure;
imshow(NewVariableName)
```

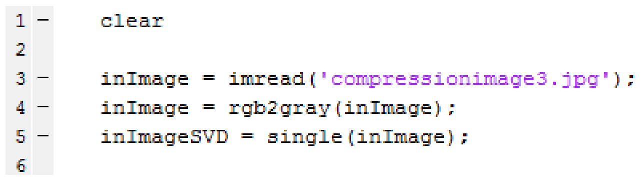


4 Applying the SVD to image compression using MATLAB

Now that we have completed exercises regarding the basic coding structures of MATLAB, we can begin the process of compressing an image.

We will begin by clearing out all previous variables to ensure that the code runs smoothly. This is done by placing the command "clear" at the beginning of the script.

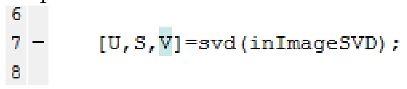
First, we will import the image that we wish to compress into MATLAB and convert it to a black and white image (see example in 3.1.2).



```
clear

inImage = imread('compressionimage3.jpg');
inImage = rgb2gray(inImage);
inImageSVD = single(inImage);

```

The "inImageSVD = single(inImage)" line of code is saving the black and white image that we imported as a matrix with values having single precision. This means that we will limit the precision of our matrix by almost half so our program will only have to work with half of the bytes of the original image. For more precision you can use the "double()" command.

The SVD command in MATLAB we will use will take our image matrix and decompose it into the three individual matrices of the SVD.



```

[U,S,V]=svd(inImageSVD);

```

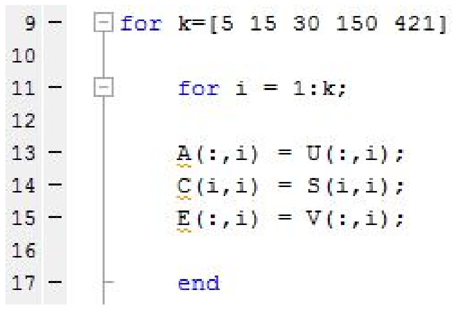
This will create the matrices U, S, and V. The matrix S represents the matrix Σ in our original SVD equation.

In our original equation, U is multiplied by Σ then $U\Sigma$ is multiplied by V^T. This means that we have to change the columns of U and the rows of V^T. This must be done by changing the number of columns of U. Since the matrix V will be transposed in the end, we will need to change the number columns of V. This will change the number rows once it has been transposed. This fulfills the required dimensions necessary for matrix multiplication.

Now that we have decomposed our matrix, we can begin to select the rank of the matrix Σ. Since Σ contains all of our singular values, choosing a rank will select how many singular values we want to have.

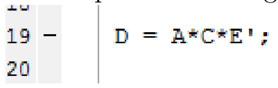
We will begin this process by applying a "for" loop where we will define our desired matrix ranks. It is best to choose between several different singular values to try, so that you can distinguish the difference in image quality. In this example, we chose ranks of 5, 15, 30, 150, and 421 (421 being the highest possible rank for this particular image). The first few matrix ranks should be spaced no more than 20 ranks apart so that you can see the gradual improvement or depreciation in quality as the rank increases or decreases. Once we hit a rank between 50 and 100, we can begin to skip a wider range (For example: 150-421).

We need to create dummy matrices that will be edited and reconstructed later on for U, S, and V in the code. This completes the process of changing matrix dimensions as was mentioned above.



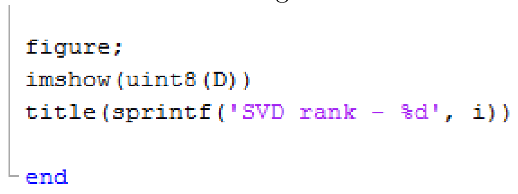
```
for k=[5 15 30 150 421]

    for i = 1:k;

    A(:,i) = U(:,i);
    C(i,i) = S(i,i);
    E(:,i) = V(:,i);

    end
```

Now we need to reconstruct the three matrices back into one by implementing the SVD equation creating a new matrix D.



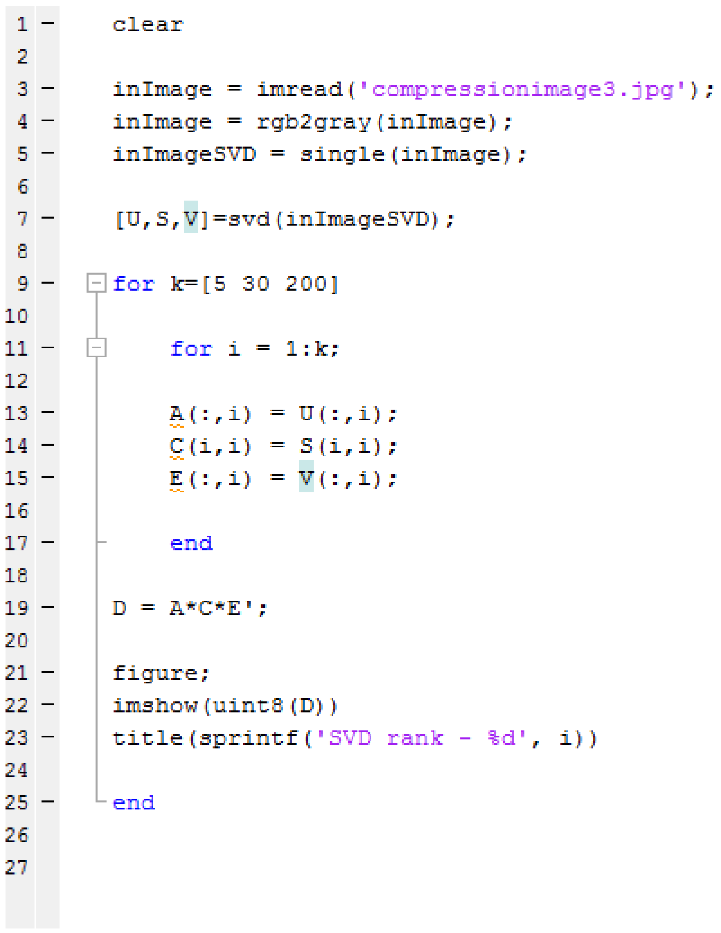
```
D = A*C*E';
```

Finally once the new image matrix D is made, we can display the images that we have chosen the singular values for.

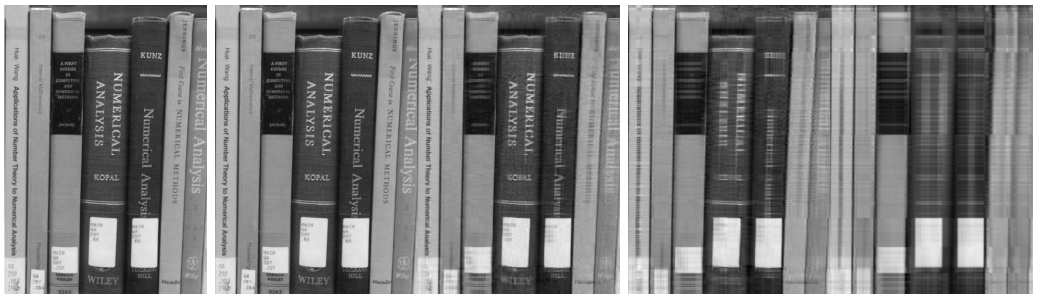


```
figure;
imshow(uint8(D))
title(sprintf('SVD rank - %d', i))

end
```

The code "title(sprintf('SVD rank - %d', i))" creates the text, "SVD rank - (number of the rank)" above the image while updating the number of the rank as it goes. Here is the code in its entirety.



```
clear

inImage = imread('compressionimage3.jpg');
inImage = rgb2gray(inImage);
inImageSVD = single(inImage);

[U,S,V]=svd(inImageSVD);

for k=[5 30 200]

    for i = 1:k;

    A(:,i) = U(:,i);
    C(i,i) = S(i,i);
    E(:,i) = V(:,i);

    end

D = A*C*E';

figure;
imshow(uint8(D))
title(sprintf('SVD rank - %d', i))

end
```



(a) rank 421 (b) rank 150 (c) rank 30 (d) rank 15 (e) rank 5

These are our resulting images. You can see how with decreasing singular values, the quality of the image depreciates and becomes blurry and jagged.

Lowering the singular values (or the rank of Σ) compresses our image. This compression decreases the amount of memory that the image takes up in storage; as you can see here, the compression of an image comes at a cost.

4.1 Memory and an Application

The memory of the original image with 421 singular values (a) turned out to be 113 KB. The last image with 5 singular values (e) had a memory value of 87 KB. From having 421 singular values going to 5 singular values, we saved about 23% of memory for this image.

23% can be a lot when you have a very limited amount of space or have to store a large amount of photos on a camera. This compression although is extremely useful when applying the concept onto smartphones. You may have a limited amount of memory, but now you have the ability to take a greater amount of pictures than before.