Math 551 Spring 2018
Lab 11

Goal: In this lab, you will learn to use the singular value decomposition for image compression.

Getting started: You will need to download the file `TarantulaNebula.png` and put it in your working directory.

What you have to submit: The file `m551lab11.m`, which you will create during the lab session.

Reminders About the Grading Software

Remember, the labs are graded with the help of software tools. If you do not follow the instructions, you will lose points. If the instructions ask you to assign a value to a variable, you must **use the variable name given in the instructions**. If the instructions ask you to make a variable named `x1` and instead you make a variable named `x` or `X` or `X1` or any name other than `x1`, the grading software will not find your variable and you *will* lose points. Required variables are listed in the lab instructions in a gray box like this:

Variables: `x1`, `A`, `q`

At times you will also be asked to answer questions in a comment in your M-file. You must **format your text answers correctly**. Questions that you must answer are shown in gray boxes in the lab. For example, if you see the instruction

Q7: What does the Matlab command `lu` do?

your file must contain a line similar to the following somewhere

```
% Q7: It computes the LU decomposition of a matrix.
```

The important points about the formatting are

- The answer is on a single line.
- The first characters on the line is the comment character `%`.
- The answer is labeled in the form `Q`n`:`, where n stands for the question number from the gray box.

If you do not format your answers correctly, the grading software will not find them and you *will* lose points.

Tasks

1. Open an M-file called `m551lab11.m` and add the following commands

   ```
   t=linspace(0,2*pi,100);
   X=[cos(t);sin(t)];
   figure(1);
   plot(X(1,:),X(2,:),'b');
   axis equal
   ```

   this will create a 2×100 matrix $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_{100}]$ whose columns are unit vectors pointing in various directions. The plot will show a blue circle corresponding to the endpoints of these vectors.

   Variables: `X`

2. Now in the M-file, define a variable `A` holding the matrix

   $$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix}$$

   and a variable `AX` holding the product `A*X`. Note that this product equals

   $$\mathbf{AX} = [\mathbf{Ax}_1\ \mathbf{Ax}_2\ \cdots\ \mathbf{Ax}_{100}],$$

   so the columns of `AX` show where the matrix $\mathbf{A}$ maps each of the columns of $\mathbf{X}$. Add the following lines to your M-file after the definition of `AX`.

```
figure(2);
plot(AX(1,:),AX(2,:),'b');
axis equal
```

Now when you run the M-file you should see a blue circle in Figure 1 and a blue ellipse in Figure 2. This shows that $\mathbf{A}$ maps vectors that end on the unit circle to vectors ending on an ellipse.

Variables: `A`, `AX`

3. Add the following line to your M-file and re-run the file.

```
[U,S,V] = svd(A)
```

Your output should look like this

```
U =

   -0.9571    0.2898
    0.2898    0.9571


S =

    2.3028         0
         0    1.3028


V =

   -0.9571   -0.2898
   -0.2898    0.9571
```

The function `svd` computes the singular value decomposition of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

where $\mathbf{U} = [\mathbf{u}_1\ \mathbf{u}_2]$ and $\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2]$ are orthogonal matrices and

$$\mathbf{S} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$$

is a non-negative diagonal matrix. Moreover

$$[\mathbf{A}\mathbf{v}_1\ \mathbf{A}\mathbf{v}_2] = \mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V} = \mathbf{U}\mathbf{S} = [\mathbf{u}_1\ \mathbf{u}_2]\begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} = [\sigma_1\mathbf{u}_1\ \sigma_2\mathbf{u}_2],$$

showing that

$$\mathbf{A}\mathbf{v}_1 = \sigma_1\mathbf{u}_1 \qquad \text{and} \qquad \mathbf{A}\mathbf{v}_2 = \sigma_2\mathbf{v}_2.$$

You can check this fact in the command window as follows.

```
>> A - U*S*V'

ans =

   1.0e-15 *

    0.8882    0.4441
   -0.4441    0.2220

>> U'*U

ans =
```

```
    1.0000         0
         0    1.0000

>> V'*V

ans =

     1     0
     0     1
```

Variables: `U`, `S`, `V`

4. Plot the columns of **V** on Figure 1 by adding the following to your M-file.

```
figure(1);
hold on;
quiver(0,0,V(1,1),V(2,1),0,'r');
quiver(0,0,V(1,2),V(2,2),0,'g');
hold off;
```

The vector corresponding to the first column will be plotted in red and the second column in green.

5. Plot the columns of **US** on Figure 2 by adding the following to your M-file.

```
figure(2);
hold on;
quiver(0,0,S(1,1)*U(1,1),S(1,1)*U(2,1),0,'r');
quiver(0,0,S(2,2)*U(1,2),S(2,2)*U(2,2),0,'g');
hold off;
```

The vector corresponding to the first column will be plotted in red and the second column in green.

6. Now we'll work on a bigger matrix. Add the following commands to your M-file and run it. (If you want, you can create a new cell by entering two percent signs (`%%`) at the beginning of a line and just run the cell from now on.)

```
%% (start a new cell)

Img = double(imread('TarantulaNebula.png'));
figure(3);
image(Img);
colormap(gray(256));
```

This code loads an image as a real matrix and displays it on the screen. Each entry in the matrix corresponds to a pixel on the screen and takes a value somewhere between 0 (black) and 255 (white).

Variables: `Img`

7. Perform the singular value decomposition of `Img`, save the output in matrices `UImg`, `SImg`, and `VImg`.

Variables: `UImg`, `SImg`, `VImg`

8. Add the following code and run the M-file.

```
figure(4);
semilogy(diag(SImg));
```

This shows the singular values (the diagonal entries of the **S** matrix) for our image matrix (the scale of the y axis is logarithmic). Notice that the diagonal entries of **S** are ordered so that $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots$. One way of writing the SVD is

$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

where r is the rank of $\mathbf{A}$ and $\mathbf{u}_i$ and $\mathbf{v}_i$ are the ith columns of $\mathbf{U}$ and $\mathbf{V}$ respectively.

If for some $k < r$, σ_{k+1} is very small compared to σ_1, we should expect

$$\mathbf{A} \approx \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T$$

to be a good approximation of $\mathbf{A}$. (This is called a truncated SVD.) This idea can be used for image compression as follows. Instead of storing the whole $m \times n$ matrix $\mathbf{A}$, we instead store the $m \times k$ and $n \times k$ matrices

$$\mathbf{C} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_k] \qquad \text{and} \qquad \mathbf{D} = [\sigma_1 \mathbf{v}_1 \ \sigma_2 \mathbf{v}_2 \ \cdots \ \sigma_k \mathbf{v}_k].$$

If $k(m+n)$ is much smaller than mn, then storing $\mathbf{C}$ and $\mathbf{D}$ will take much less space than storing $\mathbf{A}$. Moreover, if we wish to display the image, we can reconstruct $\mathbf{A}$ (approximately) as $\mathbf{A} \approx \mathbf{C}\mathbf{D}^T$.

9. Add the following code to your M-file.

```
%% (start a new cell)

k = 10;

% compressed image
C = UImg(:,1:k);
D = VImg(:,1:k)*SImg(1:k,1:k);

% compression percentage
pct = 1 - (numel(C)+numel(D))/numel(Img);

figure(5);
image(C*D');
colormap(gray(256));
title( sprintf( '%.2f%% compression', 100*pct ) );

figure(6);
image(Img);
colormap(gray(256));
title( 'Original' );
```

This code compresses the image as described above using $k = 10$ singular values, then displays the reconstructed image along with the original. It also shows the percent decrease in storage size required for the compressed image. Try several values of k to see how the quality and compression percentage vary with k. Set `k` so that the compression percentage is 80.23% before turning in your M-file.

Variables: `k`, `C`, `D`