Project 14: Singular Value Decomposition and image compression

Goals: In this project we will discuss a singular value decomposition of a matrix and its applications to image compression and noise filtering.

To get started:

- Download the file `lab14.m` and put it in your working directory[19].
- Download the files `einstein.jpg`[20] and `checkers.pgm`[21] and put them in your working directory.

Matlab commands used: `clear, linspace, subplot, quiver, plot, hold on... hold off, svd, axis, title, imread, imshow, size, min, for... end, cos, sin, ', rand, double, ones`

What you have to submit: The file `lab14.m` which you will modify during the lab session.

INTRODUCTION

Any $m \times n$ matrix $\mathbf{A}$ possesses a singular value decomposition of the form:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

where $\mathbf{U}$ is an orthogonal $m \times m$ matrix satisfying the condition $\mathbf{U}^T\mathbf{U} = \mathbf{I}_m$, $\mathbf{\Sigma}$ is an $m \times n$ rectangular diagonal matrix with nonnegative values $\sigma_1, \sigma_2, \ldots, \sigma_{min(m,n)}$ on the main diagonal, and $\mathbf{V}$ is an orthogonal $n \times n$ matrix satisfying the condition $\mathbf{V}^T\mathbf{V} = \mathbf{I}_n$. The nonnegative numbers $\sigma_1, \sigma_2, \ldots, \sigma_{min(m,n)}$ are called the singular values of the matrix $\mathbf{A}$. They are arranged in the decreasing order: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{min(m,n)} \geq 0$. Observe that the singular value decomposition of a matrix is not unique.

Singular value decomposition is important for implementation of many numerical algorithms in linear algebra. In this project we will look at the geometric meaning of singular value decomposition and two applications related to image processing and noise filtering. Observe that more efficient image compression methods exist. The examples in this project are used mainly to show the reduction in the amount of data (so-called dimension reduction) which can be accomplished with singular value decomposition. Dimension reduction is an important tool in many practical applications dealing with large amounts of data, such as statistics, data science, and machine learning.

TASKS

1. We will start with an illustration of the geometric meaning of singular value decomposition. Let us look at a singular value decomposition of a 2×2 matrix. Open the file `lab16.m`, locate the code cell `%% 2x2 matrix`, and add the following commands

[19] The printout of the file `lab14.m` can be found in the appendices of this book.

[20] The image is available at: `https://commons.wikimedia.org/wiki/File:Albert_Einstein_Head.jpg`

[21] Image from page 446 of "The standard Hoyle; a complete guide and reliable authority upon all games of chance or skill now played in the United States, whether of native or foreign introduction" (1909), image appears on Flickr Commons.

```
%% 2x2 matrix
clear;
t=linspace(0,2*pi,100);
X=[cos(t);sin(t)];
subplot(2,2,1);
hold on;
plot(X(1,:),X(2,:),'b');
quiver(0,0,1,0,0,'r');
quiver(0,0,0,1,0,'g');
axis equal
title('Unit circle')
hold off;
```

This code will create a 2×100 matrix $\mathbf{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_{100}]$ whose columns $\mathbf{x}_i$ are unit vectors pointing in various directions. The plot will show a blue circle corresponding to the endpoints of these vectors and two vectors of the standard basis on the plane $e_1 = (1,0)^T$ and $e_2 = (0,1)^T$. We use the function `subplot` which will create a plot containing four subplots arranged in two rows and two columns. The plot above will occupy the first "cell" of this plot. Observe the command `quiver` which draws a vector with the beginning point given by the first two arguments (`(0,0)` in this case) and an ending point given by the next two arguments (`(1,0)` or `(0,1)` in the code above).

Variables: `X`

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

```
A = [ 2, 1; -1, 1 ];
```

and compute the singular value decomposition of this matrix using the `svd` function:

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

Using the workspace window of the Matlab main environment, check out the matrices `U`, `S`, `V`. Perform the commands

```
U'*U
V'*V
```

to ascertain that the matrices `U` and `V` are orthogonal. The output should produce 2×2 identity matrices.

Variables: `A, U, S, V`

3. Next, let us observe the geometric meaning of the individual matrices $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$ (`U, S, V` in our Matlab code) in the singular value decomposition. To do this, observe the transformations induced by these matrices on a unit circle and the vectors of the standard basis $\mathbf{e}_1$, $\mathbf{e}_2$. Let us start by multiplying the coordinates of the points of the circle and the vectors $\mathbf{e}_1$, $\mathbf{e}_2$ by the matrix $\mathbf{V}$. Execute the following code:

```
VX=V'*X;
subplot(2,2,2)
hold on;
plot(VX(1,:),VX(2,:),'b');
quiver(0,0,V(1,1),V(1,2),0,'r');
quiver(0,0,V(2,1),V(2,2),0,'g');
axis equal
title('Multiplied by matrix V^T')
hold off;
```

Observe that the matrix `VX` contains the vectors of the matrix `X` transformed by the multiplication by the matrix $\mathbf{V}^T$. Since the matrices $\mathbf{V}$ and $\mathbf{V}^T$ are orthogonal, multiplication by the matrix $\mathbf{V}^T$ is equivalent to rotation of a plane, possibly in combination with a reflection along some straight line. This allows us to conjecture that the image of the unit circle under this mapping will still be a unit circle, but the vectors of the basis will be rotated and possibly switched in orientation.
Variables: `VX`
Q1: Did the multiplication by the transpose of the matrix `V` resulted in a reflection of the plane?

4. Now, let us multiply the result from the previous step by the matrix $\mathbf{\Sigma}$ (`S` in the Matlab code). Observe that since the matrix $\mathbf{\Sigma}$ is diagonal, then multiplication by this matrix geometrically means stretching of the plain in two directions. To verify this, execute the following Matlab code:

```
SVX = S*VX;
subplot(2,2,3);
hold on;
plot(SVX(1,:),SVX(2,:),'b');
quiver(0,0,S(1,1)*V(1,1),S(2,2)*V(1,2),0,'r');
quiver(0,0,S(1,1)*V(2,1),S(2,2)*V(2,2),0,'g');
axis equal
title('Multiplied by matrix \Sigma V^T')
hold off;
```

Observe that, as expected, the unit circle is stretched and becomes an ellipsis. The images of the standard basis vectors are stretched as well.
Variables: `SVX`

5. Finally, multiply the results from the last step by the matrix $\mathbf{U}$ to obtain:

```
AX = U*SVX;
subplot(2,2,4)
hold on;
plot(AX(1,:),AX(2,:),'b');
quiver(0,0,U(1,1)*S(1,1)*V(1,1)+U(1,2)*S(2,2)*V(1,2),U(2,1)*S(1,1)*V(1,1)+...
    U(2,2)*S(2,2)*V(1,2),0,'r');
quiver(0,0,U(1,1)*S(1,1)*V(2,1)+U(1,2)*S(2,2)*V(2,2),U(2,1)*S(1,1)*V(2,1)+...
    U(2,2)*S(2,2)*V(2,2),0,'g');
axis equal
title('Multiplied by matrix U\Sigma V^T=A')
hold off;
```

Observe that the result is equivalent to multiplying the initial vector `X` by the matrix `A`. Since the matrix `U` is orthogonal, then the multiplication by this matrix should result in a rotation of the plane possibly combined with a reflection. Confirm this by observing the images of the basis vectors.
Variables `AX`
Q2: Did multiplication by the matrix `U` produce a reflection of the plane?

6. If you answered yes to both Q1 and Q2 above, can you modify the matrices `U` and `V` in such a way that no reflections of the plane occur? Produce the modified matrices `U1` and `V1` and confirm that `U1*S*V1'=A`. Observe that this shows that singular value decomposition

is not unique.
Variables `U1`, `V1`

7. Finally, observe that

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

showing that

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

Check this fact numerically by computing the expression `A*V-U*S`.

8. Now we will look at image compression using SVD. Add the following commands to your M-file:

```
%% Image compression
clear;
ImJPG=imread('einstein.jpg');
figure;
imshow(ImJPG);
[m,n]=size(ImJPG);
```

This code loads an image as a `uint8` 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: `ImJPG`

9. Perform a singular value decomposition of the matrix `ImJPG` and save the output in matrices `UIm`, `SIm`, and `VIm`. Since `ImJPG` is integer-valued, you will need to use `svd(double(ImJPG))`.
Variables: `UIm`, `SIm`, `VIm`

10. Plot the singular values using the code:

```
figure;
plot(1:min(m,n),diag(SIm));
```

This shows the singular values (the diagonal entries of the `SIm` matrix) for the image matrix `ImJPG`. Notice that the diagonal entries of `SIm` are ordered in the decreasing order $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{min(m,n)} \geq 0$.

11. The idea behind compression of data with singular value decomposition is in the following. Using the singular value decomposition, the matrix $\mathbf{A}$ can be written in the form:

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

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.

Observe that since the singular values are arranged in the decreasing order, the first terms in this sum provide a larger contribution to the matrix $\mathbf{A}$ then the subsequent terms. It may happen that for some $k < r$, σ_{k+1} is small compared to σ_1, and correspondingly does not affect the matrix $\mathbf{A}$ too much. We should then 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 to the matrix $\mathbf{A}$ (this is called a truncated SVD). In fact, there is a theorem in linear algebra which states that SVD truncated to the k terms is the best approximation to the matrix $\mathbf{A}$ among the matrices of the rank at most k in the sense of Frobenius norm (which is equivalent to the regular Euclidean norm if we consider the matrix $\mathbf{A}$ as an mn-dimensional vector).

This idea can be used for image compression as follows. Instead of storing the whole $m \times n$ matrix $\mathbf{A}$, we can 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 is much smaller than $min(m, n)$, 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 the approximation of $\mathbf{A}$ as $\mathbf{A} \approx \mathbf{C}\mathbf{D}^T$.

Add the following code to your M-file:

```
% Approximation of the original matrix with 50, 100, and 150 singular values
for k=50:50:150
    ImJPG_comp=uint8(UIm(:,1:k)*SIm(1:k,1:k)*(VIm(:,1:k))');
    figure;
    imshow(ImJPG_comp)
    % compression percentage
    pct = 1 - (numel(UIm(:,1:k))+numel(VIm(:,1:k)*SIm(1:k,1:k)))/numel(ImJPG);
    fprintf('Compression percentage for %2.0f singular values: %8.3f\n',k, pct);
end;
```

This code compresses the image as described above using $k = 50$, $k = 100$, and $k = 150$ singular values and displays the reconstructed image. Compare the reconstructed images with the original. The code also displays the compression percentage as `pct`.

Variables: `ImJPG_comp`, `pct`

12. Observe that the singular value decomposition can also be used to smooth noisy data, especially if the data contains patterns. Data smoothing is often necessary because all measurements contain small errors resulting in a "noise". This noise usually determines the smallest singular values of the matrix. Dropping these small values, thus, not only saves the storage space, but also allows to eliminate noise from the data.

    Start a new code cell. Load the file `checkers.pgm` into Matlab and add some noise to the resulting image matrix using the following code:

    ```
    %% Noise filtering
    clear;
    ImJPG=imread('checkers.pgm')
    [m,n]=size(ImJPG);
    ImJPG_Noisy=double(ImJPG)+50*(rand(m,n)-0.5*ones(m,n));
    figure;
    imshow(ImJPG);
    figure;
    imshow(uint8(ImJPG_Noisy));
    ```

    Variables: `ImJPG`, `ImJPG_Noisy`

13. Compute the SVD of the matrix `ImJPG_Noisy` and save the resulting decomposition matrices as `UIm`, `SIm`, and `VIm`.

    Variables: `UIm`, `SIm`, `VIm`

14. Compute the approximations of the initial image with $k = 10$, $k = 30$, and $k = 50$ singular values. Display the resulting approximations and compare then to the "noisy" image. Observe that SVD significantly reduces the noise. Compare the images to the initial image without noise. Observe also that even though SVD reduces the noise, it also somewhat blurs the image.

Printout of the file `lab14.m`

```
%% 2x2 matrix

clear;
t=linspace(0,2*pi,100);
X=[cos(t);sin(t)];
subplot(2,2,1);
hold on;
plot(X(1,:),X(2,:),'b');
quiver(0,0,1,0,0,'r');
quiver(0,0,0,1,0,'g');
axis equal
title('Unit circle')
hold off;

A = [ 2, 1; -1, 1 ];
[U,S,V] = svd(A);
U'*U
V'*V

VX=V'*X;
subplot(2,2,2)
hold on;
plot(VX(1,:),VX(2,:),'b');
quiver(0,0,V(1,1),V(1,2),0,'r');
quiver(0,0,V(2,1),V(2,2),0,'g');
axis equal
title('Multiplied by matrix V^T')
hold off;

SVX = S*VX;
subplot(2,2,3);
hold on;
plot(SVX(1,:),SVX(2,:),'b');
quiver(0,0,S(1,1)*V(1,1),S(2,2)*V(1,2),0,'r');
quiver(0,0,S(1,1)*V(2,1),S(2,2)*V(2,2),0,'g');
axis equal
title('Multiplied by matrix \Sigma V^T')
hold off;

AX = U*SVX;
subplot(2,2,4)
hold on;
plot(AX(1,:),AX(2,:),'b');
quiver(0,0,U(1,1)*S(1,1)*V(1,1)+U(1,2)*S(2,2)*V(1,2),U(2,1)*S(1,1)*V(1,1)+...
    U(2,2)*S(2,2)*V(1,2),0,'r');
quiver(0,0,U(1,1)*S(1,1)*V(2,1)+U(1,2)*S(2,2)*V(2,2),U(2,1)*S(1,1)*V(2,1)+...
    U(2,2)*S(2,2)*V(2,2),0,'g');
axis equal
title('Multiplied by matrix U\Sigma V^T=A')
hold off;

%% Modified SVD
```

```
U1(:,1)=U(:,1);
U1(:,2)=-U(:,2);
V1(:,1)=V(:,1);
V1(:,2)=-V(:,2);

U1*S*V1'-A

%% Check
A*V-U*S

%% Image compression

% Creates a two-dimensional array with the dimensions equal to the dimensions of
% the image
clear;
ImJPG=imread('einstein.jpg');
figure;
imshow(ImJPG);

[m,n]=size(ImJPG);

% Compute an SVD

[UIm,SIm,VIm]=svd(double(ImJPG));

% plot the singular values
figure;
plot(1:min(m,n),diag(SIm));

%% Create approximations to the image

% With 50, 100, and 150 singular values
for k=50:50:150
    ImJPG_comp=uint8(UIm(:,1:k)*SIm(1:k,1:k)*(VIm(:,1:k))');
    figure;
    imshow(ImJPG_comp)
    % compression percentage
    pct = 1 - (numel(UIm(:,1:k))+numel(VIm(:,1:k)*SIm(1:k,1:k)))/numel(ImJPG);
    fprintf('Compression percentage for %2.0f singular values: %8.3f\n',k, pct);
end;

%% Noise filtering
clear;
ImJPG=imread('checkers.pgm')
[m,n]=size(ImJPG);

% Add some noise to the image
ImJPG_Noisy=double(ImJPG)+50*(rand(m,n)-0.5*ones(m,n));
figure;
imshow(ImJPG);

figure;
imshow(uint8(ImJPG_Noisy));
```

```
[UIm,SIm,VIm]=svd(ImJPG_Noisy);

figure;
plot(1:min(m,n),diag(SIm),'ko');

for k=10:20:50
    ImJPG_comp=uint8(UIm(:,1:k)*SIm(1:k,1:k)*(VIm(:,1:k))');
    figure;
    imshow(ImJPG_comp)
    % compression percentage
    pct = 1 - (numel(UIm(:,1:k))+numel(VIm(:,1:k)*SIm(1:k,1:k)))/numel(ImJPG);
    fprintf('Compression percentage for %2.0f singular values: %8.3f\n',k, pct);
end;
```

References

[1] S. Attaway. *Matlab: A practical Introduction to Programming and Problem solving.* Butterworth-Heinemann, 3rd edition, 2013.

[2] K. Bryan, T. Leise. *The $25,000,000,000 Eigenvector: The Linear Algebra behind Google.* SIAM Rev., **48(3)**, 569 - 581, 2006.

[3] T. Chartier. *When Life is Linear: From Computer Graphics to Bracketology.* Mathematical Association of America, 2015.

[4] W.N. Colley. *Colley's bias free college football ranking method: The Colley matrix explained.* www.Colleyrankings.com, 2002.

[5] L. Elden. *Numerical linear algebra in data mining.* Acta Numerica, **15**, 327-384, 2006.

[6] C. Hennig, M. Meila, F. Murthagh and R. Rocci (eds.). *Handbook of Cluster Analysis.* CRC Press, 2015.

[7] J. Keener. *The Perron–Frobenius Theorem and the Ranking of Football Teams.* SIAM Rev., **35(1)**, 80-93, 1993.

[8] A.N. Langville, C.D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings.* Princeton University Press, 2006.

[9] J. Leskovec, R. Sosic. *SNAP: A General-Purpose Network Analysis and Graph-Mining Library,.* ACM Transactions on Intelligent Systems and Technology (TIST), **8(1)**, 2016.

[10] F.M. Harper, J.A. Konstan, *The MovieLens Datasets: History and Context.* ACM Transactions on Interactive Intelligent Systems (TiiS), **5(4)**, 2015.