SVD Example: Image compression

Author: Seung-Sup Lee

We will provide a visual understanding of how the SVD can be used to compress large matrices. First load a sample image data (*Frauenkirche* and *Neues Rathaus* in *Marienplatz*; © Thomas Wolf, www.foto-tw.de; retrieved from https://commons.wikimedia.org/wiki/File:Stadtbild_M%C3%BCnchen.jpg).

```
clear

M = imread('Marienplatz.jpg'); % Read image data from a picture
```

When `imread` does not work (e.g., for Octave), simply load it from .mat file:

```
load('Marienplatz.mat');
```

To see the information of variables, type:

```
whos M
```

```
  Name          Size                 Bytes  Class    Attributes

  M          3285x4861            15968385  uint8
```

If you use graphic interface, just look the workspace panel which is usually on the upper-right corner of the MATLAB main window. We see that `M` is 3285 $\times$ 4861 matrix of `uint8` type. `uint8` is unsigned integer each occupying 8 bit (= 1 byte) of memory. Therefore, `M` is about 15.2 MB!

Visualization of matrix

MATLAB provides several functions to visualize matrices.

```
figure; % open new figure window
imshow(M); % display image with matrix of uint8 type (NOT double).
```

`imshow` does not work for double type variables (which are generally used in MATLAB calculations). So for general purpooses, use `imagesc` as below.

```
figure;
imagesc(M); % display image with axes.
colormap(gray); % set colormap as gray, since the picture is black-and-white.
```

Note that the height-to-width ratio of pictures by `imshow` is the same as the original picture, but the ratio of pictures by `imagesc` is fitted to the figure window size.

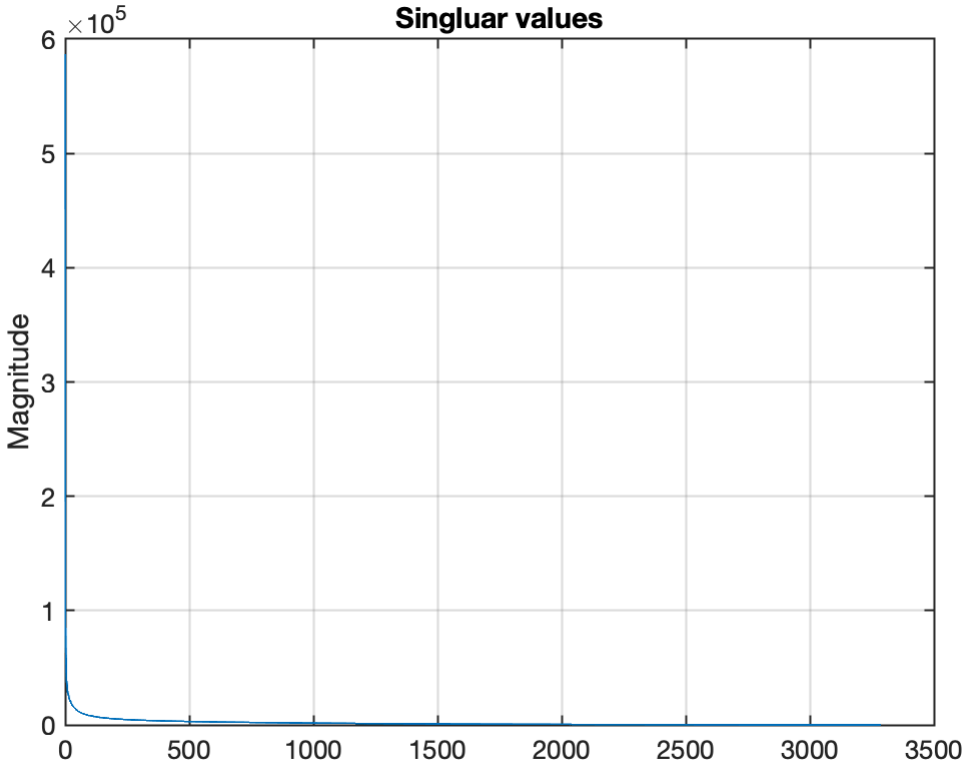
SVD of picture data

To use the SVD, we need to convert `M` to `double` type variable.

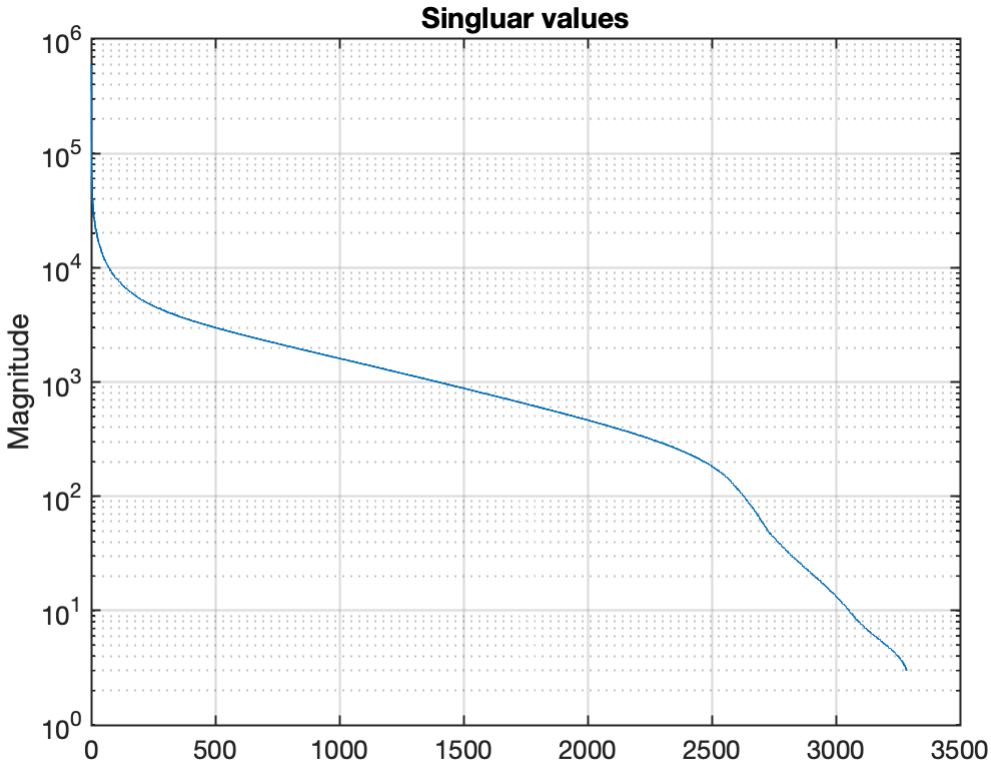
```
M2 = double(M); % convert data type: uint8 -> double
[U,S,V] = svd(M2); % singular value decomposition
```

To see the distribution of the singular values, we plot them.

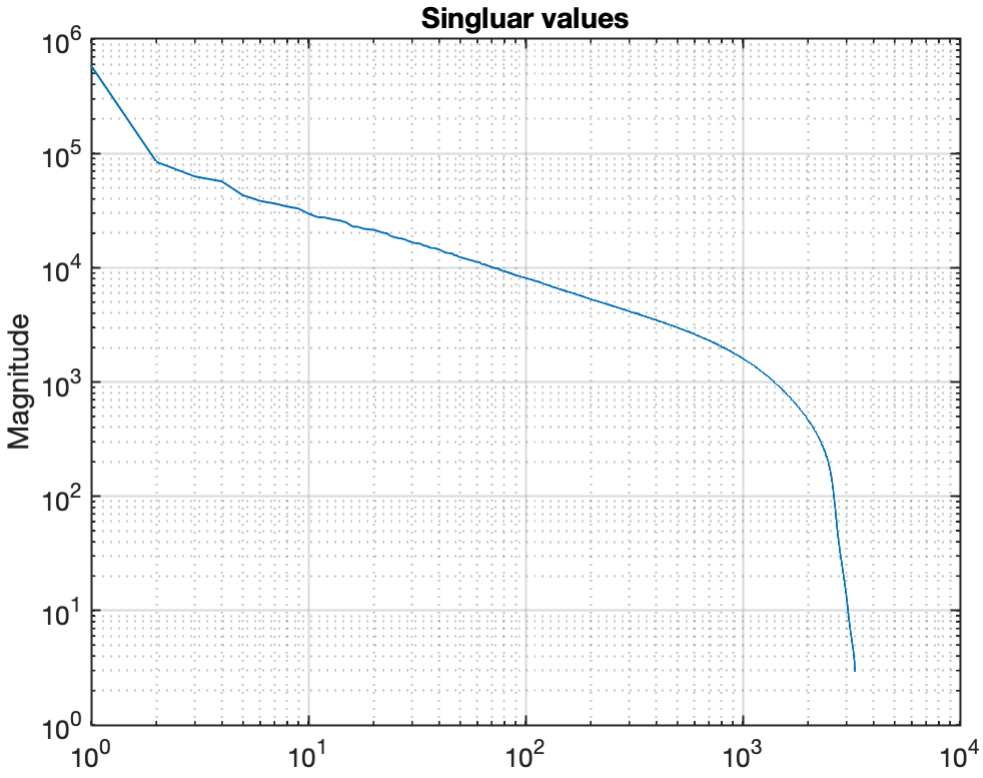
```
% The magnitude of the singluar values decays exponentially.
figure;
plot(diag(S),'LineWidth',1); % plot diagonal elements of S.
set(gca,'LineWidth',1,'FontSize',13)
title('Singluar values'); % add title
ylabel('Magnitude'); % add y-axis label
grid on; % turn on grid line
```




```
% To better see the exponential decay, plot in log-linear scale
figure;
semilogy(diag(S),'LineWidth',1); % plot diagonal elements of S.
set(gca,'LineWidth',1,'FontSize',13)
title('Singluar values'); % add title
ylabel('Magnitude'); % add y-axis label
grid on; % turn on grid line
```




```
% plot in log-log scale
figure;
loglog(diag(S),'LineWidth',1); % plot diagonal elements of S.
set(gca,'LineWidth',1,'FontSize',13)
title('Singluar values'); % add title
ylabel('Magnitude'); % add y-axis label
grid on; % turn on grid line
```

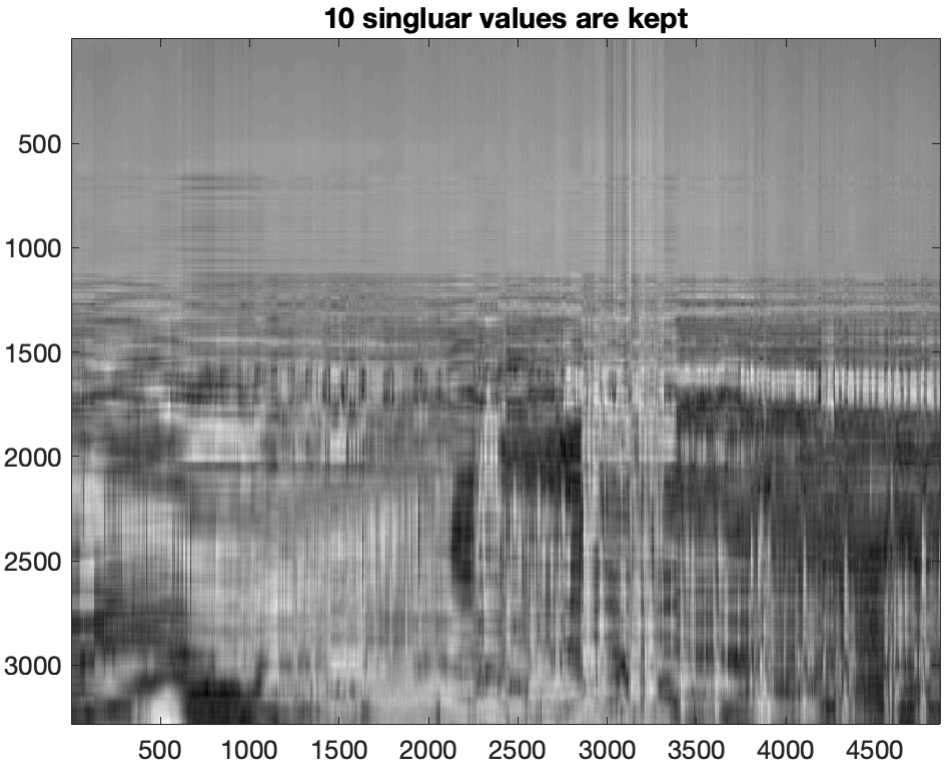


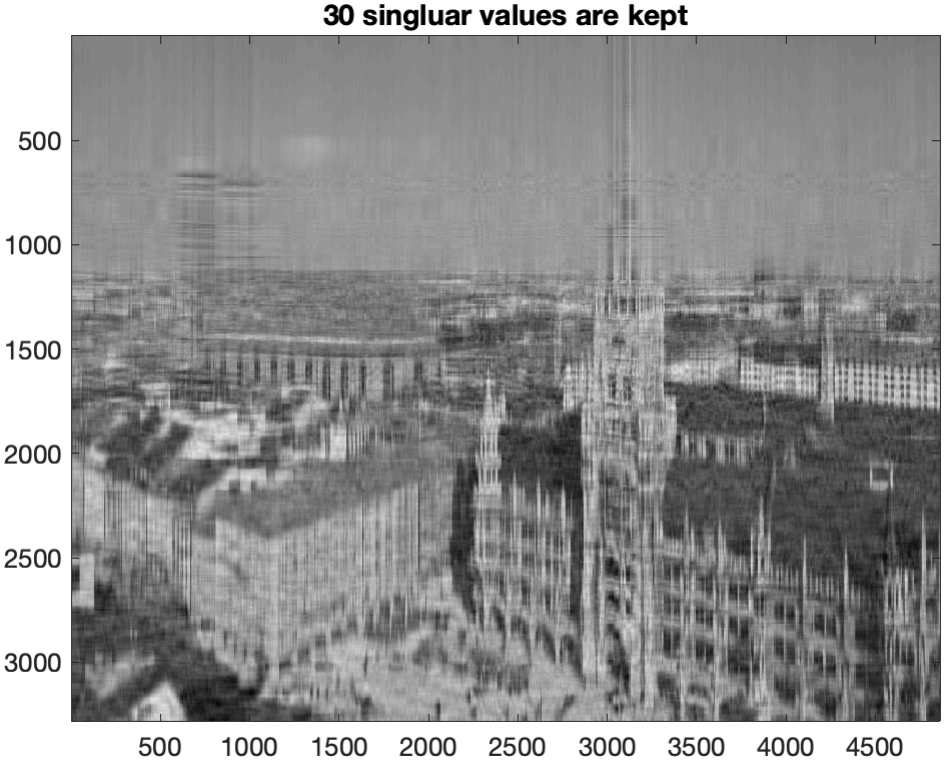
Reconstruction of picture

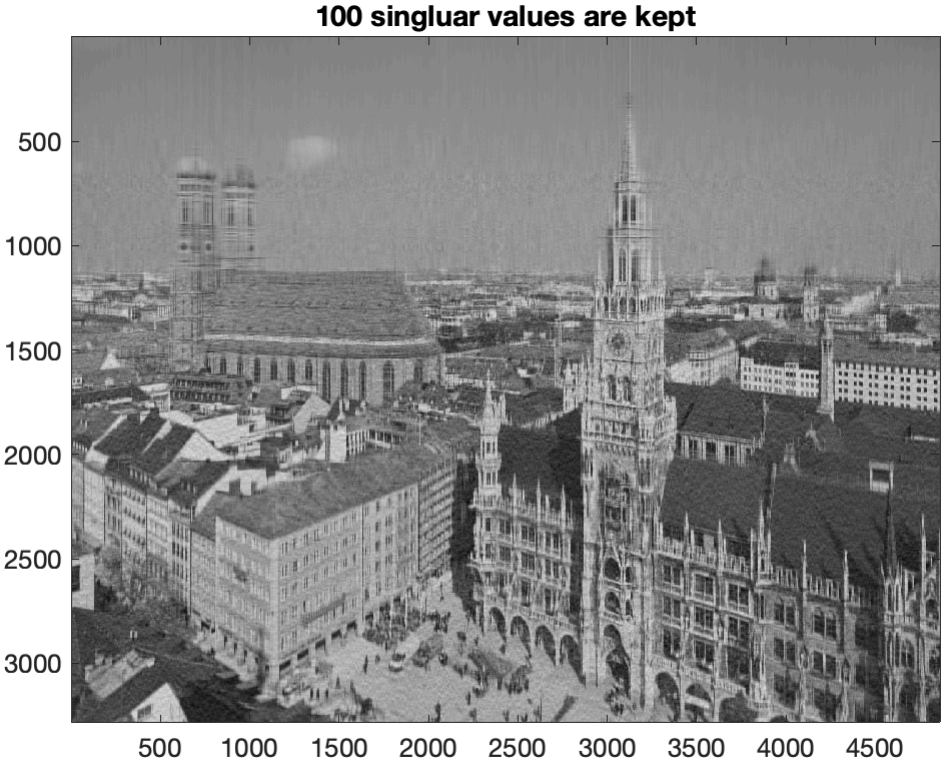
Now we reconstruct picture from the SVD result of `M`. It is clear that `U*S*V'` will return the same matrix as `M` (up to double precision $\sim$1e-16). But what if we use only part of `U`, `S`, and `V`? Based on the exponential decay of the singular values, we can think of an approach that keeps only some of the largest singular values and the corresponding singular vectors.

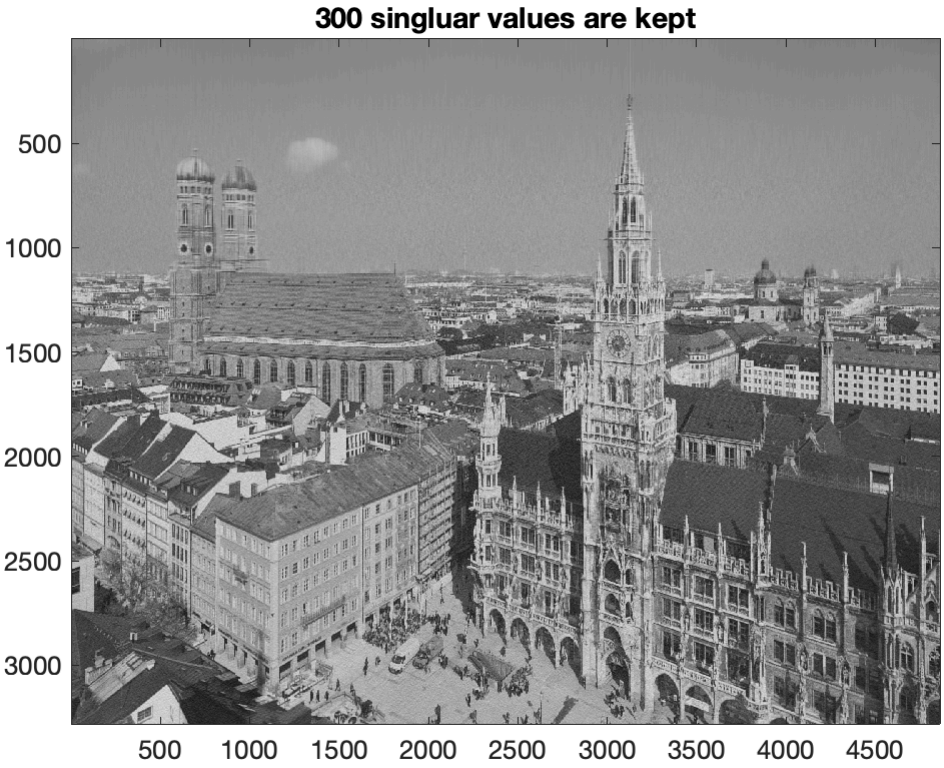
Let's compare how pictures will look like with different number of kept singular values.

```
Nkeep = [10,30,100,300]; % different number of singular values to keep

Ms = cell(numel(Nkeep),1); % cell array to contain matrices

for it = (1:numel(Nkeep))
    Ms{it} = U(:,1:Nkeep(it))*S(1:Nkeep(it),1:Nkeep(it))*V(:,1:Nkeep(it))';

    figure;
    imagesc(Ms{it});
    colormap(gray);
    set(gca,'FontSize',13)
    title([sprintf('%i',Nkeep(it)),' singluar values are kept']);
end
```


10 singluar values are kept
500
1000
1500
2000
2500
3000
500
1000
1500
2000
2500
3000
3500
4000
4500


30 singluar values are kept
500
1000
1500
2000
2500
3000
500
1000
1500
2000
2500
3000
3500
4000
4500





Only with 30 singular values, the rough shapes of *Frauenkirche* and *Neues Rathaus* are already visible. With 300 singular values (about 9% of total singular values), we can reproduce the overall features of the original

high-resoultion picture nicely! Of course, if you zoom in, you will realize that sharp details, such as the steeple of *Neues Rathaus,* remain noisy somehow.

Exercise (a): Understanding singular vectors

From the demonstration above, we have found that the singular vectors for the largest singular values (e.g. `U(:,(1:10))` and `V(:,(1:10))`) contribute more to the original matrix `M` than the singular vectors for the smallest singular values (e.g. `U(:,(end-9:end))` and `V(:,(end-9:end))`). Can you find the qualitative differences between the vectors for the largest singular values and the vectors for the smallest singular values? Use `fft` (Fast Fourier transform) for analyzing the vectors. The exercise is designed to make students familiar with reading and understanding MATLAB documentation. If you didn't read the documentation for `fft`, please read it through to the end.