

Tutorial 7: Singular value decomposition

In this Tutorial you will learn how to implement the singular value decomposition in Matlab and how to use it to compress images.

Singular value decomposition

Recall from class that the singular value decomposition (SVD) of an $m \times n$ matrix \mathbf{A} is

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

where the an $m \times n$ matrix

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

with \mathbf{D} being a diagonal $r \times r$ matrix whose r diagonal elements are the singular values

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r,$$

and where \mathbf{U} is a $m \times m$ orthogonal matrix ($\mathbf{U}^T\mathbf{U} = \mathbf{I}$), and where \mathbf{V} is an $n \times n$ orthogonal matrix ($\mathbf{V}^T\mathbf{V} = \mathbf{I}$).

In Matlab, you can compute the SVD of a matrix \mathbf{A} with the command `svd`. Use help to find out how to use it.

```
help svd
```

SVD Singular value decomposition.

`[U,S,V] = SVD(X)` produces a diagonal matrix S , of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that $X = U*S*V'$.

`S = SVD(X)` returns a vector containing the singular values.

`[U,S,V] = SVD(X,0)` produces the "economy size" decomposition. If X is m -by- n with $m > n$, then only the first n columns of U are computed and S is n -by- n . For $m \leq n$, `SVD(X,0)` is equivalent to `SVD(X)`.

`[U,S,V] = SVD(X,'econ')` also produces the "economy size" decomposition. If X is m -by- n with $m \geq n$, then it is equivalent to `SVD(X,0)`. For $m < n$, only the first m columns of V are computed and S is m -by- m .

See also `SVDS`, `GSVD`.

Reference page in Doc Center
`doc matlab/svd`

Other functions named `svd`

`codistributed/svd` `gpuArray/svd` `sym/svd` `tall/svd`
`frd/svd`

Now you are ready to use it. Use the `svd` command to compute the svd of the matrix

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

```
U = 3x3
    -0.3333    0.6667   -0.6667
     0.6667    0.6667    0.3333
    -0.6667    0.3333    0.6667

S = 3x2
    4.2426     0
         0     0
         0     0

V = 2x2
    -0.7071    0.7071
     0.7071    0.7071
```

You can check that $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$:

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

```
ans = 3x2
1e-15 x
    0.3331   -0.3331
   -0.4441    0.4441
    0.4441   -0.4441
```

Note, as usual, the numerical error. The matrix $\mathbf{A} - \mathbf{U}^*\mathbf{S}^*\mathbf{V}'$ should be equal to the zero matrix, but because of numerical error, it is a matrix with small elements.

Reduced SVD, the pseudo inverse and low-rank approximations

Recall from class that the rank of a matrix \mathbf{A} is the number of non-zero singular values. Since the zero singular values do not really contribute to \mathbf{A} , one can construct a more compact singular value decomposition by skipping the zero singular values. This also means that one skips columns of \mathbf{U} and \mathbf{V} . For an $m \times n$ matrix with rank r , define

$$\mathbf{U}_r = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_r],$$

$$\mathbf{V}_r = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r],$$

where \mathbf{U}_r is $m \times r$ and \mathbf{V}_r is $n \times r$. In other words, use only the left and right singular vectors corresponding to non-zero singular vectors only. You can check that

$$\mathbf{A} = \mathbf{U}_r \mathbf{D} \mathbf{V}_r^T.$$

This more compact version of the singular value decomposition is called the *reduced singular vector decomposition*.

The matrix \mathbf{A} we defined above has rank 1 (only one singular value is non-zero). We can thus compute the reduced singular vector decomposition by

```
Ur = U(:,1); % use only the first column of U
Vr = V(:,1); % use only the first column of V
D = S(1,1); % construct the r x r matrix D that has the nonzero singular values on its diagonal
```

Check that $\mathbf{A} = \mathbf{U}_r \mathbf{\Sigma}_r^T \mathbf{V}_r$:

```
A - Ur*D*Vr'
```

```
ans = 3x2
10^-15 x
    0.3331   -0.3331
   -0.4441    0.4441
    0.4441   -0.4441
```

You can see again the numerical error: $\mathbf{U}_r \mathbf{\Sigma}_r^T \mathbf{V}_r$ is equal to \mathbf{A} up to the numerical error.

It is common to encounter matrices that have no zero singular values, but many which are nearly zero. The matrix stored in the file "SVD_Example.mat" is an example of that. You can load this file with the load command. If you forgot how to use it, use help:

```
clear % delete old variables, we don't need them anymore
help load
```

LOAD Load data from MAT-file into workspace.
S = LOAD(FILENAME) loads the variables from a MAT-file into a structure array, or data from an ASCII file into a double-precision array. Specify FILENAME as a character vector or a string scalar. For example, specify FILENAME as 'myFile.mat' or "myFile.mat".

S = LOAD(FILENAME, VARIABLES) loads only the specified variables from a MAT-file. Specify FILENAME and VARIABLES as character vectors or string scalars. When specifying VARIABLES, use one of the following forms:

VAR1, VAR2, ...	Load the listed variables. Use the '*' wildcard to match patterns. For example, load('A*') loads all variables that start with A.
'-regex', EXPRESSIONS	Load only the variables that match the specified regular expressions. For more information on regular expressions, type "doc regexp" at the command prompt.

S = LOAD(FILENAME, '-mat', VARIABLES) forces LOAD to treat the file as a MAT-file, regardless of the extension. Specifying VARIABLES is optional.

S = LOAD(FILENAME, '-ascii') forces LOAD to treat the file as an ASCII file, regardless of the extension.

LOAD(...) loads without combining MAT-file variables into a structure array.

LOAD ... is the command form of the syntax, for convenient loading from the command line. With command syntax, you do not need to enclose inputs in single or double quotation marks. Separate inputs with spaces instead of commas. Do not use command syntax if FILENAME is a variable.

Notes:

If you do not specify `FILENAME`, the `LOAD` function searches for a file named `matlab.mat`.

ASCII files must contain a rectangular table of numbers, with an equal number of elements in each row. The file delimiter (character between each element in a row) can be a blank, comma, semicolon, or tab. The file can contain MATLAB comments.

Examples:

```
gongStruct = load('gong.mat')      % All variables
load('handel.mat', 'y')           % Only variable y
load('accidents.mat', 'hwy*')     % Variables starting with "hwy"
load('topo.mat', '-regex', '\d')  % Variables containing digits

% Using command form
load gong.mat
load topo.mat -regex \d
load 'hypothetical file.mat'      % Filename with spaces
```

See also `SAVE`, `MATFILE`, `WHOS`, `CLEAR`, `REGEXP`, `IMPORTDATA`, `UIIMPORT`.

Reference page in Doc Center
`doc load`

Other functions named `load`

COM/load

Now you can load the file:

```
load('./Data/SVD_Example.mat')
```

And compute the SVD of the 356 x 356 matrix **A**

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

Recall that **S** is a 356 x 356 matrix (same size as **A**), but the singular values of **A** are on its diagonal. You can get the diagonal elements of **S** via the `diag` command. Use help to find out how to use it:

```
help diag
```

DIAG Diagonal matrices and diagonals of a matrix.

`DIAG(V,K)` when `V` is a vector with `N` components is a square matrix of order `N+ABS(K)` with the elements of `V` on the `K`-th diagonal. `K = 0` is the main diagonal, `K > 0` is above the main diagonal and `K < 0` is below the main diagonal.

`DIAG(V)` is the same as `DIAG(V,0)` and puts `V` on the main diagonal.

`DIAG(X,K)` when `X` is a matrix is a column vector formed from the elements of the `K`-th diagonal of `X`.

`DIAG(X)` is the main diagonal of `X`. `DIAG(DIAG(X))` is a diagonal matrix.

Example

```
m = 5;
diag(-m:m) + diag(ones(2*m,1),1) + diag(ones(2*m,1),-1)
produces a tridiagonal matrix of order 2*m+1.
```

See also `SPDIAGS`, `TRIU`, `TRIL`, `BLKDIAG`.

Reference page in Doc Center
doc diag

Other functions named diag

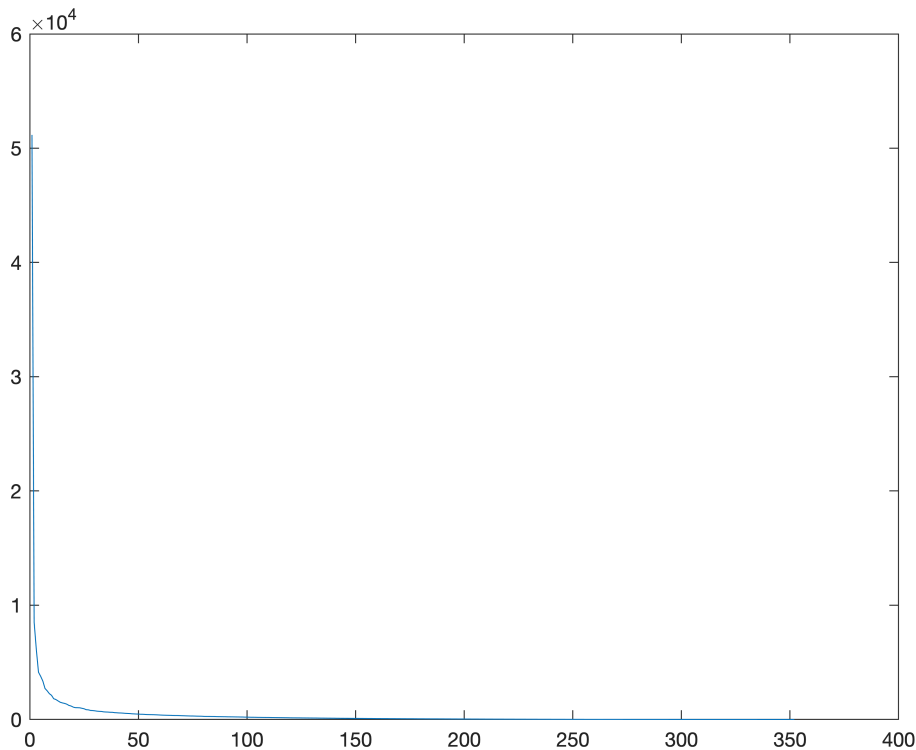
`codistributed/diag` `gpuArray/diag` `sym/diag` `umat/diag`

Now you are ready to use the command and define a 356 x 1 vector **Sd** whose elements are the singular values of **A**:

```
Sd = diag(S);
```

You can see that many singular values are small (compared the the largest singular value) by plotting the singular values:

```
figure  
plot(Sd)
```



You can see that the singular values past the 50th one are relatively small. This means that you might be able to neglect them without making large errors. Neglecting the small singular values means replacing them by zero (since they are small, you might as well set them to zero). In this way, you can use the reduced singular value decomposition to construct a matrix $\hat{\mathbf{A}}$ that is very nearly equal to the matrix **A**. The procedure is as follows. You select how many singular values you want to keep. Here, we want to keep 50. Then define:

$$\hat{\mathbf{U}} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{50}],$$

$$\hat{\mathbf{V}} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{50}],$$

and define the 50 x 50 diagonal matrix $\hat{\mathbf{D}}$ which has the 50 largest singular values of \mathbf{A} on its diagonal. Then define the matrix $\hat{\mathbf{A}}$ by its reduced singular value decomposition:

$$\hat{\mathbf{A}} = \hat{\mathbf{U}}\hat{\mathbf{D}}\hat{\mathbf{V}}^T.$$

In Matlab, you implement this step as follows:

```
Dhat = diag(Sd(1:50));
Uhat = U(:,1:50);
Vhat = V(:,1:50);

Ahat = Uhat*Dhat*Vhat';
```

Note that \mathbf{A} has rank 356 (it has 356 non-zero singular values) and $\hat{\mathbf{A}}$ has rank 50. Since the rank of $\hat{\mathbf{A}}$ is lower than the rank of \mathbf{A} , the matrix $\hat{\mathbf{A}}$ is called a low-rank approximation.

To see how good this approximation is, we revisit where the matrix \mathbf{A} actually comes from. The matrix \mathbf{A} is in fact an image. More precisely, it contains the intensity of the red channel of a color image of a snowboarder. To see the image, we need to convert the elements of the matrix back to what is used in imaging. You do this via the command `uint8` (but the details are not relevant for linear algebra).

```
A_image = uint8(A);
```

Now you can also plot the image using the `imagesc` command:

```
figure
imagesc(A_image)
colormap bone
```



You can compare the image to its low-rank approximation using the same commands:

```
Ahat_image = uint8(Ahat);  
figure  
imagesc(Ahat_image)  
colormap bone
```



You can see that the low-rank approximation is pretty good in the sense that you can still clearly see a snowboarder, but if you look closely, you will notice that the low-rank approximation is a bit more pixelated than the original image, i.e., you lost some detail in the low-rank approximation.

Exercise

The image of the snowboarder is not black and white, it is a color image. This means that it consists of three matrices, one for the red, one for the green and one for the blue channel. These matrices are stored in the file "SVD_Excercise.mat". Load the file via the load command (use help if you forgot how to use it).

```
clear % delete old variables
load('./Data/SVD_Excercise.mat')
```

You now have three matrices, A_red, A_green, A_blue. To make an image out of the the three channles, use the following comands that store all three matrices in one data strucutre.

```
% convert matrices to image file strucutre
A_rgb(:,:,1) = A_red;
A_rgb(:,:,2) = A_green;
A_rgb(:,:,3) = A_blue;
A_rgb = uint8(A_rgb);
```

You can now plot the color image:

```
figure
imagesc(A_rgb)
```




By modifying the code from the tutorial, compute low-rank approximations of the three matrices, A_{red} , A_{green} , A_{blue} . It is up to you how many singular values you want to keep. Call the resulting three matrices $A_{\text{red_lr}}$, $A_{\text{green_lr}}$ and $A_{\text{blue_lr}}$.

Solution

Low-rank approximation of the red channel:

```
[U_red, S_red, V_red] = svd(A_red);
S_red = diag(S_red);

keepRed = 20; % how many singular values to keep

Ulr_red=U_red(:,1:keepRed);
Slr_red = diag(S_red(1:keepRed));
Vlr_red = V_red(:,1:keepRed);
A_red_lr = Ulr_red*Slr_red*Vlr_red';
```

Now the green channel

```
[U_green, S_green, V_green] = svd(A_green);
S_green = diag(S_green);

keepGreen = 20; % how many singular values to keep
```

```

Ulr_green=U_green(:,1:keepGreen);
Slr_green = diag(S_green(1:keepGreen));
Vlr_green = V_green(:,1:keepGreen);
A_green_lr = Ulr_green*Slr_green*Vlr_green';

```

Now the blue channel

```

[U_blue, S_blue, V_blue] = svd(A_blue);
S_blue = diag(S_blue);

keepBlue = 20; % how many singular values to keep

Ulr_blue=U_blue(:,1:keepBlue);
Slr_blue = diag(S_blue(1:keepBlue));
Vlr_blue = V_blue(:,1:keepBlue);
A_blue_lr = Ulr_blue*Slr_blue*Vlr_blue';

```

Then you can use the following code to assemble and plot the color image.

```

A_rgb_lr(:, :, 1) = A_red_lr;
A_rgb_lr(:, :, 2) = A_green_lr;
A_rgb_lr(:, :, 3) = A_blue_lr;
A_rgb_lr = uint8(A_rgb_lr);

figure
imagesc(A_rgb_lr)

```

