Image compression based on the singular value decomposition

Introduction

We shall look at the *singular value decomposition* (SVD) of a matrix A, which is gone through in Section 12.3 in Sauer. You are not required to read this in order to solve this project. Rather the purpose here is to analyse what kind of information the SVD captures, and illustrate how it provides a method for compressing images by reducing storage requirements.

The main result in Section 12.3 in Sauer says that, if A is an $m \times n$ nonzero, real matrix, it can be factored as

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

where U and V are orthogonal matrices (i.e., $U^{-1} = U^T$, $V^{-1} = V^T$) of dimensions $m \times m$ and $n \times n$, respectively, and where Σ is an $m \times n$ "diagonal matrix" on the form

$$\Sigma = \left(\begin{array}{cccc|cccc} \sigma_1 & 0 & \dots & \dots & \dots & \dots & \cdots & 0 \\ 0 & \sigma_2 & 0 & \dots & \dots & \dots & \cdots & 0 \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & 0 & \sigma_r & 0 & \vdots & \vdots & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

where r is the rank of A (written rank(A)), and where the diagonal elements $\sigma_1, \dots, \sigma_r$ are the nonzero *singular values* of A. All these are positive and ordered in such a way that

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0.$$

The factorization $A = U\Sigma V^T$ described above is called the *SVD* of A.

Exercise 1

Let $A = U\Sigma V^T$ be the SVD of the $m \times n$ matrix A, and let $\mathbf{u}_j$ and $\mathbf{v}_j$ be the j-th column of U and V, respectively, i.e.,

$$U = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \end{pmatrix} \qquad V = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{pmatrix}.$$

a)

Show that

$$U\Sigma = \begin{pmatrix} \sigma_1\mathbf{u}_1 & \sigma_2\mathbf{u}_2 & \cdots & \sigma_r\mathbf{u}_r & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}$$

where the number of zero-vectors is $n - r$.

Hint: Look at the element at the position i, j in the matrix $(U\Sigma)$ and use SVD. Then collect all elements in column j.

b)

Recall that if B is an $m \times p$ matrix, and C is an $p \times n$ matrix, then the matrix product BC can be written as

$$BC = \sum_{j=1}^{p} \texttt{col}_j(B)\,\texttt{row}_j(C).$$

Use this and **a)** to show that

$$A = \sum_{j=1}^{r} \sigma_j\, \mathbf{u}_j\mathbf{v}_j^T. \tag{1}$$

Comment: **b)** shows that only the r first columns of U and V matter in the SVD $A = U\Sigma V^T$. Note also that all terms in the sum (1) are rank 1 matrices, since $\mathbf{u}\mathbf{v}^T = [v_1\mathbf{u} \quad v_2\mathbf{u} \quad \ldots \quad v_n\mathbf{u}]$, so that $\texttt{rank}(\mathbf{u}\mathbf{v}^T) = 1$ when $\mathbf{u}$ and $\mathbf{v}$ are both nonzero. **b)** thus also shows that a rank r matrix A can be written as a sum of r rank one matrices.

Some useful Python commands

We will consider an application of the SVD to image analysis. An image can simply be considered as a matrix, a rectangular grid where each point in the grid corresponds to an element in the matrix. In image analysis one rather uses the term *pixel*, instead of grid point.

In *gray tone* images every pixel has an associated value measuring its intensity. Pixel values in such images are often stored as integers between 0 and 255, where 0 corresponds to black and 255 corresponds to white. *Colour images* on the other hand are often stored in the *rgb-format*, where every pixel has three components, one for red (r), one for green (g), and one for blue (b). All possible colours can be produced by combining these three base colours in different ways. This means that we need 3 different matrices to store an image in the rgb format. In this project we shall stick to gray tone images, so that one matrix is enough to store the image.

Below is a description of some Python commands that may be useful. A is here a two-dimensional `numpy` array or matrix. It is assumed that the following imports have been made:

```
import matplotlib.pyplot as plt
from numpy import *
```

The `linalg.matrix_rank` function in `numpy` returns the rank of A, and can be invoked in two ways:

```
r=linalg.matrix_rank(A)
r=linalg.matrix_rank(A,tol=epsilon)
```

The second line here requires some explanation. We are used to finding the rank by row reducing the matrix, and counting the number of pivot columns. The entries in a matrix are sensitive to roundoff errors, however. The effect of these can be that, after row reduction, it may look like the matrix has full rank, even if this is not the case. Instead we can use the fact that the rank equals the number of singular values greater than 0. Python applies this procedure, and assumes that $\sigma_j > 0$ if $\sigma_j > \epsilon$ for a given tolerance $\epsilon > 0$. This tolerance is set on the second line above (`tol=epsilon`).

The function `U,S,V = linalg.svd(A)` in the `numpy` module returns the singular value decomposition of the A, i.e., it returns matrices U, V, and a vector S so that $A = U\Sigma V$, where Σ is diagonal with the entries of S on the diagonal. Note the difference from the mathematical definition in that the matrix V is not transposed!

We also have the following three functions in the `matplotlib.pyplot` module

```
A = double(plt.imread(filename,format=None))
plt.imwrite(filename,A,format=str)
plt.imshow(img,cmap='gray')
```

On the first line an image is read from a given file in a given format, and stored it in the matrix A. Note the conversion to `double` here, which is needed. On the second line the image stored in A is written to a given file, with the given format (to print images you have generated, use this command first, then send the resulting file to the printer). On the third line the matrix A is displayed as an image. The named parameter `cmap='gray'` states that the image should be interpreted as a grayscale image (with values between 0 and 255).

Exercise 2

We continue using the notation from Exercise 1. For $1 \leq k \leq r$ we define

$$A^{(k)} = \sum_{j=1}^{k} \sigma_j \mathbf{u_j} \mathbf{v_j}^T. \tag{2}$$

$A^{(k)}$ has rank k and can thus be thought of as a rank k approximation to A. How many terms we need to include, i.e., how big k must be for an acceptable approximation, will of course depend on the singular values, and how quickly they tend to 0.

a)

Write a function `svdApprox(A,k)` which takes an $m \times n$ matrix A, and k, as input, computes the SVD of the matrix, and returns the matrix $A^{(k)}$ in (2). If $k > r = \text{rank}(A)$ the function should provide a suitable error message.

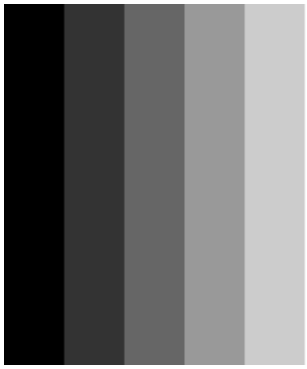

Figure 1: An image with a particularly simple pattern

b)

Load the Marilyn Monroe image mm.gif provided in this project into a matrix A. Check that A is a 256×256 matrix, and compute its rank with help of the `linalg.matrix_rank`-function. Also compute the rank with tolerances 0.001, 0.01, 0.1, and 1.

c)

Use the function you implemented in **a)** to display the image corresponding to $A^{(k)}$ for $k = 8$ og $k = 32$.

d)

Since the rank of the image matrix equals the number of nonzero singular values, a low rank means that we can represent the matrix with just a few values. Can you say anything about the rank of the matrix in Figure 1, just by looking at the image?

Exercise 3

The singular values are listed in decreasing order with the biggest first. Thus, the most "important" information is found in the first terms in the sum (1).

a)

Plot the singular values of the matrix from the Marilyn Monroe image (σ_j as a function of j).

b)

Construct a random image of the same size, and with pixel values in the same interval as the Marilyn Monroe image. This you can do by writing

```
B = (255*random.rand(256,256)).round()
```

Plot the singular values of the matrix of the random image together with the singular values of the matrix of the Marilyn Monroe image. Compare how the singular values decrease for the two matrices, and try to explain what you see.

Exercise 4

Every term in the sum (2) needs two vectors, in addition to one singular value. To store the matrix $A^{(k)}$ we therefore only need to store the k first vectors in both matrices U and V, as well as the k first singular values.

a)

How many numbers do you need to store a rank k SVD approximation to an image of size $m \times n$?

b)

How big k is needed in order for the rank k SVD approximation to be visually as good as the original? Experiment with Python. Try different grayscale images, also of larger size.

c)

The *relative error* between the matrices A and $A^{(k)}$ is defined as

$$e_{rel} = \frac{\|A - A^{(k)}\|}{\|A\|},$$

where the (Frobenius) norm of an $m \times n$ matrix $B = [b_{i,j}]$ is defined by

$$\|B\| = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} {b_{i,j}}^2}.$$

Implement a function `relError(A,AK)` which computes the relative error between the two matrices `A` and `AK`. What is the relative error between the original image and the approximation you chose in **b)**?

d)

in Exercise **2c)** you reconstructed the Marylin Monroe image from the 8 first singular components, i.e., from $A^{(8)}$. Try now to reconstruct the same image by using all singular values *except from* the 8 first singular components, i.e., by using the matrix $\sum_{j=9}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T$, where $r = \text{rank}(A)$. Look at both images, and compute the relative error for each of them. Is the "visual" error in both images acceptable?

Extra exercise

We continue using the notation from Exercise 1.

a)

Let $1 \leq k \leq r$. Show that $\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$ is a basis for Col $A^{(k)}$. This shows in particular that $A^{(k)}$ has rank k (as mentioned in Exercise 2). Note that this basis is orthonormal.

b)

What is the SVD of A^T ? Display the image corresponding to A^T when A is the Marylin Monroe image. Describe the matrix corresponding to the image where Marylin has been rotated med 90 degrees counterclockwise.

c)

Explain that $\{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$ is a basis for Row A = Col A^T. Note that this basis is orthonormal.

Extra computational exercise: can we do better? (For those who are very much into programming)

The technique described in this Challenge efficiently reduces image size while keeping the image recognizable. One may pose a few further research questions, for example, whether one could use less parameters (=reduced image size) to get the same image quality, or get better image quality using the same number of parameters.

A paper from 2021 in Computers & Electrical Engineering, "Singular vector sparse reconstruction for image compression",
`https://doi.org/10.1016/j.compeleceng.2021.107069`,
suggests that one may discard some information from the computed singular vectors and still achieve comparable image quality. The authors call this technique "sampling", and Fig. 6 gives a good idea of what they are doing: take a singular vector, that is, a column of V-matrix, plot vector elements against index, and find "a bit special" elements marked by squares in the Figure. Store only those special elements in a new "sparse singular vector", and discard the rest of the original singular vector. They call the procedure for "sparse sampling".

Very Extra exercise

Implement the sampling method described in the paper and compare it with standard dense SVD compression method, using the same number of parameters.