# Lecture 11: Algorithms for computing eigenvalues and eigenvectors and Principal Component Analysis

Francesco Preta

August 2020

In this lecture we will discuss different algorithms to compute eigenvalues and eigenvectors and then illustrate an application of singular value decomposition in Principal Component analysis.

## The power method

The power method algorithm for a square $n \times n$ matrix $A$ with real eigenvalues returns the highest eigenvalue of $A$ along with the corresponding eigenvector. The algorithm is based on the premise that the highest eigenvalue of $A$ is real and has multiplicity one. In fact, if $A$ is diagonalizable, then for any $\mathbf{x} \in \mathbb{R}^n$ we can write

$$A^k\mathbf{x} = c_1\lambda_1^k\mathbf{v}_1 + c_2\lambda_2^k\mathbf{v}_2 + ... + c_n\lambda_n^k\mathbf{v}_n$$

where $\mathbf{v}_1, ..., \mathbf{v}_n$ are the eigenvectors corresponding to the eigenvalues $\lambda_1, ..., \lambda_n$ and $c_1, ..., c_n$ are the coordinates of the original vector $\mathbf{x}$ in the basis of eigenvectors. Then as $k$ grows,

$$A^k\mathbf{x} \sim c_1\lambda_1^k\mathbf{v}_1$$

Therefore if we divide the expression by $\lambda_1^k$, we get

$$\frac{A^k\mathbf{x}}{\lambda_1^k} = c_1\mathbf{v}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\mathbf{v}_2 + ... + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\mathbf{v}_n \tag{1}$$

so that $\frac{A^k\mathbf{x}}{\lambda_1^k} \to c_1\mathbf{v}_1$. Under this premise, the algorithm works as follows:

1. Select an initial vector $\mathbf{x}_0$ whose largest coordinate is 1.
2. For $k = 0, 1, ...$:
   (a) Compute $A\mathbf{x}_k$.
   (b) Let $\mu_k$ be the entry of $A\mathbf{x}_k$ with largest absolute value.
   (c) Let $\mathbf{x}_{k+1} = \frac{\mathbf{x}_k}{\mu_k}$

3. If $\mathbf{x}_0$ is not an eigenvector for another eigenvalue, the sequence $\{\mu_k\}$ approaches $\lambda_1$ and the sequence $\mathbf{x}_k$ approaches $\mathbf{v}_1$.

**Example:** consider the matrix $A = \begin{bmatrix} 6 & 5 \\ 1 & 2 \end{bmatrix}$ and $\mathbf{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Applying the algorithm for the first three iterations gives:

$$\begin{aligned}
A\mathbf{x}_0 &= \begin{bmatrix} 5 \\ 2 \end{bmatrix} & \mu_0 &= 5 & \mathbf{x}_1 &= \begin{bmatrix} 1 \\ 0.4 \end{bmatrix} \\
A\mathbf{x}_1 &= \begin{bmatrix} 8 \\ 1.8 \end{bmatrix} & \mu_1 &= 8 & \mathbf{x}_2 &= \begin{bmatrix} 1 \\ 0.225 \end{bmatrix} \\
A\mathbf{x}_2 &= \begin{bmatrix} 7.125 \\ 1.45 \end{bmatrix} & \mu_2 &= 7.125 & \mathbf{x}_3 &= \begin{bmatrix} 1 \\ 0.2035 \end{bmatrix}
\end{aligned}$$

so it seems that $\mathbf{x}_k$ converges to $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0.2 \end{bmatrix}$.

In this case we have that $\lambda_1 = 7$ and $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 0.2 \end{bmatrix}$. By diagonalizing the matrix, we find that $\lambda_2 = 1$, so that the ratio $\frac{\lambda_2}{\lambda_1} = \frac{1}{7}$ and the convergence is quite fast. In fact, the convergence depends on the ratio $\frac{\lambda_2}{\lambda_1}$ since that constitutes the main source of error between $\frac{A^k\mathbf{x}}{\lambda_1^k}$ and $c_1\mathbf{v}_1$ as can be seen by Equation (1). This hints to the reason behind the requirement of $\lambda_1$ having multiplicity one, or otherwise the eigenvector produced by the algorithm would converge slowly and produce a choice strongly dependent on the algorithm itself (for instance, different scalings could give different outputs).

Finally, notice that if we choose an input vector $\mathbf{x}_0 = c_1\mathbf{v}_1 + ... + c_n\mathbf{v}_n$ with $c_1 = 0$, then the algorithm would not converge to $\mathbf{v}_1$. However the probability of doing so by chance is very close to 0, so we can ignore such possibility.

## The inverse power method

A generalization of the power method that allows to find more than just one eigenvalue and eigenvector is the inverse power method. It's based on the assumptions that $A$ is a matrix with all different real eigenvalues $\lambda_1, ..., \lambda_n$, each different from 0. Then, since $A$ is invertible, $A^{-1}$ has eigenvalues $\lambda_1^{-1}, ..., \lambda_n^{-1}$. Let $\mathbf{v}_i$ be the eigenvector of $A$ for $\lambda_i$. Then by left multiplication by $\lambda_i A$

$$A^{-1}\mathbf{v}_i = \lambda_i^{-1}\mathbf{v}_i \iff \lambda_i\mathbf{v}_i = A\mathbf{v}_i$$

In the inverse power method, we don't know exactly what the eigenvalues are, but we have an estimation for each of them, for instance $\alpha_1, ..., \alpha_n$. Then the matrix $A - \alpha_i I$ will have an eigenvalue $\lambda_i - \alpha_i$ close to 0, since

$$(A - \alpha_i I)\mathbf{v}_i = (\lambda_i - \alpha_i)\mathbf{v}_i$$

and we started by the assumption that $\alpha_i$ and $\lambda_i$ are close to each other. Then, as long as $\alpha_i$ is not exactly equal to $\lambda_i$, it makes sense to consider the matrix

$B = (A - \alpha_i I)^{-1}$, which will have a very large eigenvalue $(\lambda_i - \alpha_i)^{-1}$. If the estimates $\alpha_1, ..., \alpha_n$ are close to the actual eigenvalues, so that $(\lambda_j - \alpha_i)^{-1}$ is significantly smaller than $(\lambda_i - \alpha_i)^{-1}$ for $j \neq i$, then the power method on $B$ converges very fast. The algorithm works in the following way:

1. Given estimates $\alpha_1, ..., \alpha_n$ to eigenvalues $\lambda_1, ..., \lambda_n$, for $i = 1, ..., n$:
   (a) Choose $\mathbf{x}_0$ having coordinate with the largest absolute value equal to 1.
   (b) For $k = 1, 2, ...$:
      i. Solve for $\mathbf{y}_{k,i}$ the equation $\mathbf{x}_{k,i} = (A - \alpha_i)\mathbf{y}_{k,i}$.
      ii. Let $\mu_{k,i}$ be the entry of $\mathbf{y}_{k,i}$ with the largest possible absolute value.
      iii. Evaluate $\nu_{k,i} = \alpha_i + \frac{1}{\mu_{k,i}}$.
      iv. Compute $\mathbf{x}_{k+1,i} = \frac{\mathbf{y}_{k,i}}{\mu_{k,i}}$
   (c) Then the sequence $\{\nu_{k,i}\}$ converges to $\lambda_i$ and the sequence $\mathbf{x}_{k,i}$ converges to the eigenvector $\mathbf{v}_i$

Notice that $B_i = (A - \alpha_i I)^{-1}$ does not appear in the algorithm. This happens because finding the largest eigenvector of the inverse is more numerically unstable than solving the equation $\mathbf{x}_{k,i} = (A - \alpha_i I)\mathbf{y}_{k,i}$.

## The QR algorithm

The two previous algorithms, power method and inverse power method, are simple tools for practical applications. Built-in methods in numpy and MATLAB generally use more sophisticated algorithms. The basis of those is generally the QR algorithm, which in the easiest formulation works as follows:

1. Consider a square matrix $A$ and let $A_0 = A$
2. For $k = 0, 1, ...,$:
   (a) Compute the QR decomposition $A_k = Q_k R_k$.
   (b) Consider $A_{k+1} = R_k Q_k$
3. The elements on the diagonal of $A_k$ are the eigenvalues of $A$.

Notice that at each iteration, $A_k$ and $A_{k+1}$ are **similar**. We have that

**Definition 1.** Two matrices $B$ and $C$ are *similar* if there exists an invertible matrix $Q$ such that

$$C = QBQ^{-1}$$

In general, every diagonalizable matrix is similar to the diagonal matrix $\Lambda$ containing its eigenvalues. Viceversa, if two matrices are similar, they have the same eigenvalues. This happens because

$$C\mathbf{v}_i = \lambda_i \mathbf{v}_i \iff QBQ^{-1}\mathbf{v}_i = \lambda_i \mathbf{v}_i \iff BQ^{-1}\mathbf{v}_i = \lambda_i Q^{-1}\mathbf{v}_i$$

so that $\lambda_i$ is an eigenvalue for $B$ with eigenvector $Q^{-1}\mathbf{v}_i$.

In this case, we have that $A_k = Q_k R_k$, so that $R_k = Q_k^{-1} A_k$. Then

$$A_{k+1} = R_k Q_k = Q_k^{-1} A_k Q_k$$

Therefore for each $k$, $A_k$ and $A_{k+1}$ are similar. That means that $A_0 = A$ and $A_k$ are similar for each $k$ and therefore they have the same eigenvalues. Under some specific conditions, $A_k$ converges to an upper triangular matrix, having eigenvalues on the principal diagonal.

This form of the QR algorithm is a good starting point, but it has many computational problems. First of all, it does not work for all the matrices. Consider $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Then it's clear that the QR decomposition has $Q = A$ and $R = I$, so that $RQ = QR = A$. Multiple iterations will not, in this case, produce an upper triangular matrix. Moreover, even if the algorithm converges, the rate of convergence to 0 of the element $A_{i,j}$ is given by $\frac{|\lambda_i|}{|\lambda_j|}$ so that if two eigenvalues are very close to each other, the corresponding element will converge very slowly. Fixing these issues with convergence and numerical stability will be the subject of a numerical analysis class.

## Principal Component Analysis

In what follows we will consider a $n \times m$ **matrix of observations** $X$. Each data point $\mathbf{x}_i$, for $i = 1, ..., n$ represents the i-th row vector of $X$ having $m$ features $\mathbf{x}_{i,j}$ for $j = 1, ..., m$.

For a given observation matrix $X$, it makes sense to define the following:

**Definition 2.** Let $X$ be an observation matrix with rows $\{\mathbf{x}_i\}_{i=1}^n$. Then the *sample mean* $\mathbf{m}$ is the vector

$$\mathbf{m} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

and the *mean-deviation form* of $X$ is the matrix

$$\hat{X} = \begin{bmatrix} \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \\ \dots \\ \hat{\mathbf{x}}_n \end{bmatrix}$$

where $\hat{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{m}$ for $i = 1, ..., n$.

**Example:** consider the observation matrix

$$X = \begin{bmatrix} 1 & 2 & 1 \\ 4 & 2 & 13 \\ 7 & 8 & 1 \\ 8 & 4 & 5 \end{bmatrix}$$

then the sample mean is given by

$$\mathbf{m} = \frac{1}{4}(\begin{bmatrix}1\\2\\1\end{bmatrix}^T + \begin{bmatrix}4\\2\\13\end{bmatrix}^T + \begin{bmatrix}7\\8\\1\end{bmatrix}^T + \begin{bmatrix}8\\4\\5\end{bmatrix}^T) = \begin{bmatrix}5 & 4 & 5\end{bmatrix}$$

so that the normalized vectors are

$$\hat{\mathbf{x}}_1 = \begin{bmatrix}-4\\-2\\-4\end{bmatrix}^T \qquad \hat{\mathbf{x}}_2 = \begin{bmatrix}-1\\-2\\8\end{bmatrix}^T \qquad \hat{\mathbf{x}}_3 = \begin{bmatrix}2\\4\\-4\end{bmatrix}^T \qquad \hat{\mathbf{x}}_4 = \begin{bmatrix}3\\0\\0\end{bmatrix}^T$$

and the observation matrix in mean-deviation form is

$$\hat{X} = \begin{bmatrix}-4 & -2 & -4\\-1 & -2 & 8\\2 & 4 & -4\\3 & 0 & 0\end{bmatrix}$$

The mean-deviation forms normalizes the observation matrix so that the vectors have zero mean. This simplifies the calculations for the covariance matrix.

**Definition 3.** Let $X$ be an observation matrix and $\hat{X}$ its mean-deviation form. Then the *sample covariance matrix* $S$ of $X$ is

$$S = \frac{1}{n-1}\hat{X}^T\hat{X}$$

The sample covariance matrix is a $m \times m$ matrix which measures the covariance between features. If two features are highly correlated, then removing one of them does not change significantly the information we have about our data. On the other hand, two features with 0 covariance are called **uncorrelated** and there is no way of knowing the value of one from the value of the other.

**Example:** the previous example gives

$$S = \frac{1}{3}\begin{bmatrix}-4 & -1 & 2 & 3\\-2 & -2 & 4 & 0\\-4 & 8 & -4 & 0\end{bmatrix}\begin{bmatrix}-4 & -2 & -4\\-1 & -2 & 8\\2 & 4 & -4\\3 & 0 & 0\end{bmatrix} = \begin{bmatrix}10 & 6 & 0\\6 & 8 & -6\\0 & -6 & 32\end{bmatrix}$$

The coordinates on the diagonal represent the **variance** of each feature. The **total variance** is the sum of all such values, given by the trace $\mathrm{Tr}(S)$. In this case $\mathrm{Tr}(S) = 50$ so that the total variance is 50.

In principal component analysis, we plan to find a change of variable $\mathbf{y} = \mathbf{x}P$ for $P$ a $m \times m$ matrix, with the property that the new features, obtained as linear combinations of the old ones, are uncorrelated and arranged in order of decreasing variance. In particular, we would like to find the matrix $P$ to be orthogonal, so that $\mathbf{y}P^T = \mathbf{x}$. In this way, we obtain $n$ observation vectors

$\{\mathbf{y}_i\}_{i=1}^n$ with coordinates corresponding to the new features, and a new observation matrix $Y$. If $X$ is in mean-deviation form (that is, if $\sum_i \mathbf{x}_i = \mathbf{0}$), then $Y$ is also in mean-deviation form, since

$$\sum_{i=1}^n \mathbf{y}_i = \sum_{i=1}^n (\mathbf{x}_i P) = (\sum_{i=1}^n \mathbf{x}_i)P = \mathbf{0}$$

Moreover, the corresponding sample covariance matrix $S_y$ is

$$S_y = \frac{1}{n-1} Y^T Y = \frac{1}{n-1} P^T X^T X P = P^T S_x P$$

Since we want to choose $P$ such that $S_y$ is a diagonal matrix with $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_m \geq 0$, we have to diagonalize $S_x$, so that the i-th column of $P$ correspond to the unit eigenvector $\mathbf{v}_i$ for the eigenvalue $\lambda_i$. Each $\mathbf{v}_i$ is called the $i$-th principal component of $X$ and identifies a linear combination of the features which captures part of the variance, while being uncorrelated with the other principal components.

**Example:** We can use a program to diagonalize the matrix $S$ from the previous example. We obtain

$$\lambda_1 = 33.416 \qquad \mathbf{v}_1 = \begin{bmatrix} 0.059 \\ 0.229 \\ -0.972 \end{bmatrix}$$

$$\lambda_2 = 10 \qquad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\lambda_3 = 6.584 \qquad \mathbf{v}_3 = \begin{bmatrix} 0.863 \\ -0.491 \\ -0.116 \end{bmatrix}$$

The coefficients of $\mathbf{v}_i$ express the linear combinations of the features that are uncorrelated to the other linear combinations. The eigenvalues represent their variance.

## Dimensionality reduction

Principal component analysis is generally used to greatly reduce the dimensionality of data. In particular, by identifying a small number of principal components containing most of the variance (and therefore, most of the information about the data points), one can project the data points to the linear subspace spanned by those new features and reduce dimensions. In our case, for

$$\hat{X} = \begin{bmatrix} -4 & -2 & -4 \\ -1 & -2 & 8 \\ 2 & 4 & -4 \\ 3 & 0 & 0 \end{bmatrix} \qquad S = \begin{bmatrix} 10 & 6 & 0 \\ 6 & 8 & -6 \\ 0 & -6 & 32 \end{bmatrix}$$

we have that the total variance is given by $\mathrm{Tr}(S) = 50$. Then we can divide each eigenvalue by the total variance to obtain

$$var_1 = \frac{\lambda_1}{\mathrm{Tr}(S)} = 66.85\% \quad var_2 = \frac{\lambda_2}{\mathrm{Tr}(S)} = 20\% \quad var_3 = \frac{\lambda_3}{\mathrm{Tr}(S)} = 13.15\%$$

so that $\lambda_1$ and $\lambda_2$ together account for approximately 86.85% of the total variance. In this way, we can project to a subspace spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$ to retain most of the information about the original data.

In order to reduce the dimensionality and have a $n \times 2$ matrix (that is, $n$ data points with two features), we can change basis $\hat{Y} = \hat{X}V$ and then consider only the first two coordinates. This can be done by considering the matrix $V_2 = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}$ and defining the new observation matrix as $\hat{Y}_2 = \hat{X}V_2$.

In the SVD decomposition, since by the definition of $S$, $\mathbf{v}_1$ and $\mathbf{v}_2$ are the unitary eigenvectors associated with the largest singular values $\sigma_1$ and $\sigma_2$,

$$\hat{Y}_2 = U\Sigma V^T V_2 = U\Sigma \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ ... & \\ 0 & 0 \end{bmatrix} = U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \\ ... & \\ 0 & 0 \end{bmatrix}$$

where $\sigma_i = \sqrt{\lambda_i}$, which is a $n \times 2$ matrix. In fact, this corresponds to first rewriting $\hat{X}$ in the coordinates given by the principal component directions and then truncating only to the first two coordinates.

In general, in order to apply dimensionality reduction, we can consider the SVD of an observation matrix in mean-deviation form $\hat{X} = U\Sigma V^T$ with covariance matrix $S$. The matrix $\hat{Y} = \hat{X}V = U\Sigma$ corresponds to a change of basis for each single data point in the coordinates given by the principal component directions. Then we can truncate up to the first $d$ principal components. This can be done either by deciding $d$ a priori, or by choosing a threshold on the relative variance. The resulting matrix $\hat{Y}_d$ is obtained by considering

$$\hat{Y}_d = \hat{X}V_d$$

where $V_d$ is the $m \times d$ matrix having columns $\mathbf{v}_i$ for $i = 1, ..., d$.