Principal Component Analysis

Naureen Ghani

March 3, 2018

Introduction

Principal components analysis (PCA) is a technique to simplify a multi-dimensional dataset to two or three dimensions., which enables data visualization. The idea is that many large datasets contain correlations between the dimensions, so that a portion of the data is redundant. PCA will transform the data so that as much variation as possible will be crammed into the fewest possible dimensions. To apply PCA, you first need to understand how the correlations between dimensions can be described by a covariance matrix.

Standard Deviation

Variance in a population is:

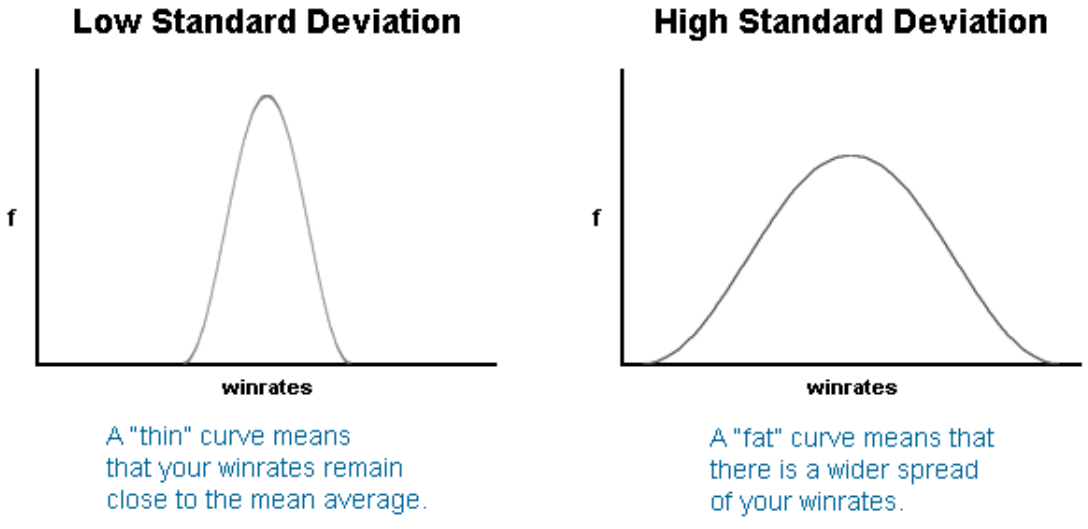
$$\sigma^2 = \frac{\sum (x - \mu)^2}{n}$$

where x is a value from the population, μ is the mean of all x, and n is the number of x in the population. The spread of a distribution is also referred to as ***dispersion*** and ***variability.***

Standard deviation is the best measure of spread in an approximately normal distribution. This means that the data is not skewed and does not contain outliers. For such datasets, the ***inter-quartile range*** (difference between the 25th and 75th percentiles) is a preferred measure of spread. SD is calculated as the square root of the variance (the average squared deviation from the mean).

Test Your Understanding:

- Suppose there exists a set of observations whose variance is 11. If each term of the set is multiplied by 3, then find the new variance of this set.



The built-in function in MATLAB to compute variance is **var.** However, it does not compute the standard deviation by taking the square root of the variance. It uses ***Bessell's correction,*** which uses n-1 instead of n in the formula for standard deviation. This is to correct for bias in the original estimate (it systematically underestimates the variance). The theory is that while there are n independent samples, there are only n-1 indepedent ***residuals.*** The ***sample variance*** is defined as:

$$s^2 = \frac{1}{n-1}\sum_{i-1}^{n}(x_i - \mu)^2$$

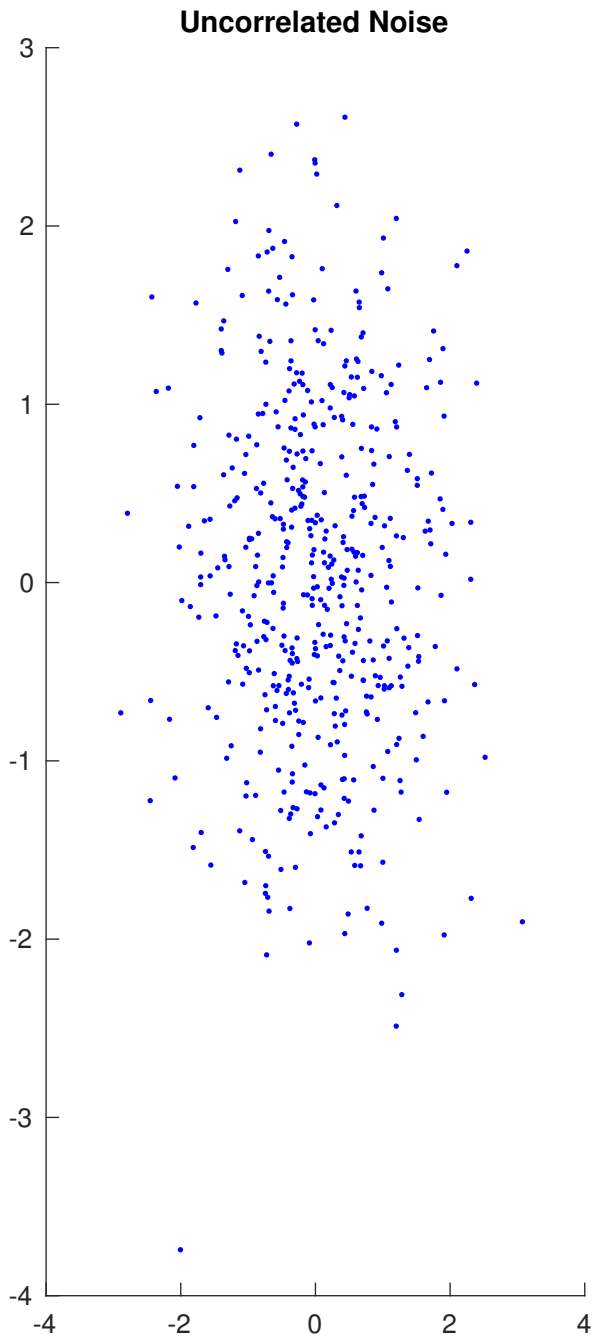
We will begin this demo with simulated data. We will generate two 2-dimensional matrices, one correlated and one uncorrelated.

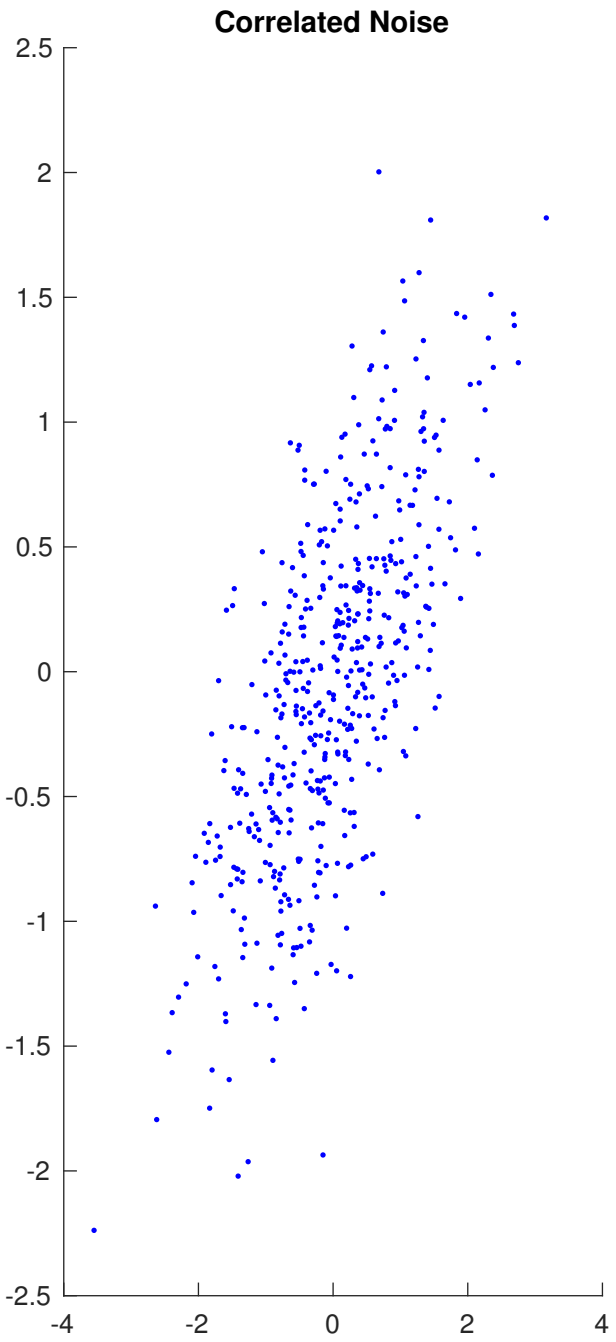
```
% Generate white noise (noise containing many frequencies with equal
% intensities

n = 500; % n = num of datapoints
a(:,1) = normrnd(0,1,n,1); % mean of 0, std of 1, n x 1 matrix
a(:,2) = normrnd(0,1,n,1); % same parameters to make 2nd dim of matrix a

b(:,1) = normrnd(0,1,n,1); % same parameters to make matrix b
b(:,2) = b(:,1)*0.5 + 0.5*normrnd(0,1,n,1); % correlate 2nd dimension with 1st
    dimension of matrix b

figure;
subplot(1,2,1); scatter(a(:,1), a(:,2), 'b','.'); title('Uncorrelated Noise');
hold on;
subplot(1,2,2); scatter(b(:,1), b(:,2), 'b', '.'); title('Correlated Noise');
```





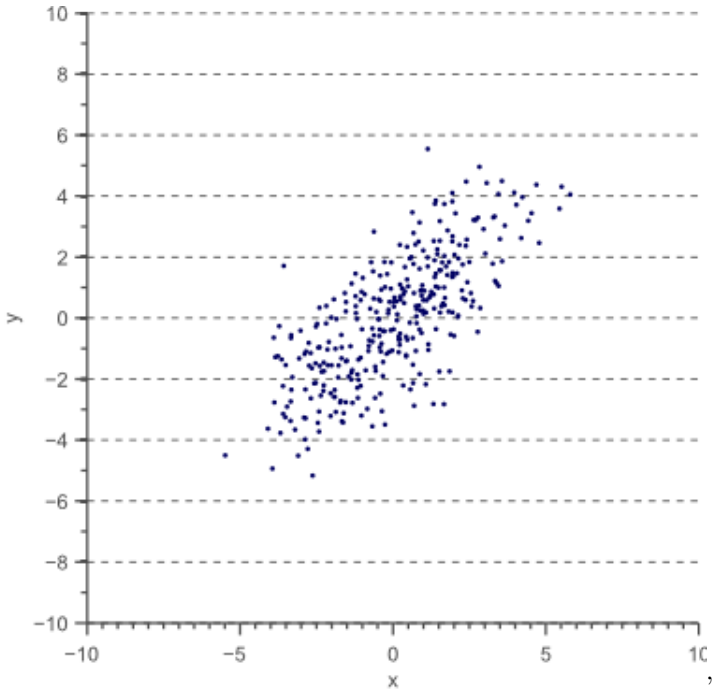
If you subtract the mean from your data, then you can more compactly express the sample variance as follows:

$$s^2 = \frac{1}{n-1}(x-\mu)^T(x-\mu)$$

The superscript T represents transpose, which means that the matrix columns are changed to rows and vice versa. In other words, an m x n matrix becomes an n x m matrix. Let us verify that the two formulas for sample variance are equivalent:

```
var(a(:,1)) % Compute the sample variance of 1st dim of "a"
c = a(:,1)-mean(a(:,1)) % Subtract mean from the 1st dim of "a"
c'*c/(n-1) % Compute sample variance of 1st dim of "a", note transpose
```

The covariance is analagous to the variance, except that is computed between two vectors, rather than a vector and itself. Variance can only explain the spread of data in the directions parallel to the axes of the ***feature space.*** Consider:



In this graph, we can calculate variance $\sigma(x,x)$ in the x-direction and variance $\sigma(y,y)$ in the y-direction. However, the horizontal spread and vertical spread of the data does not explain the diagonal spread of the data. For this, we can use the covariance.

With a second data vector y with n indepedent values, the ***sample covariance*** is:

$$cov(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$$

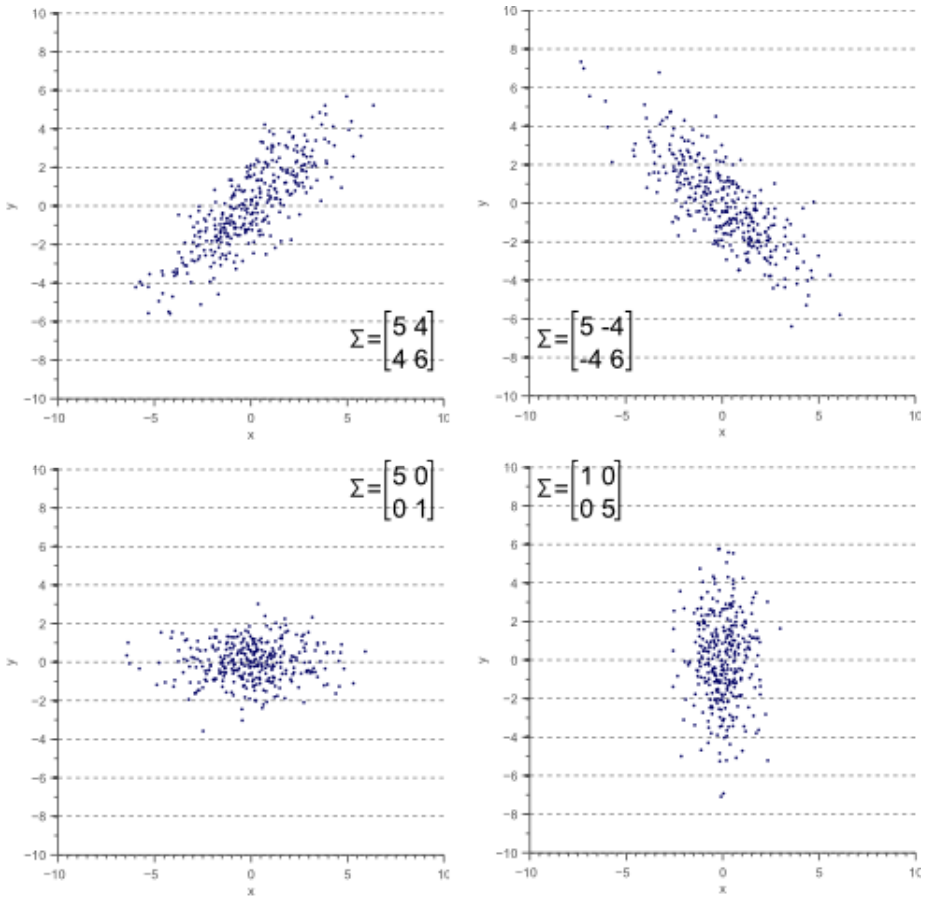
If x and y are the same, the sample covariance is identical to the sample variance. Other possibles include:

- If x and y are **uncorrelated,** the **covariance is 0.**
- If x and y are both large, there is a **positive covariance.**
- If x is large and y is small, there is a **negative covariance.**

For 2-D data, the covariance matrix is:

$$\sum = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix}$$

If x is positively correlated with y, then y is also positively correlated with x. In other words, $\sigma(x,y) = \sigma(y,x)$. Thus, the covariance matrix is always a symmetric matrix with the variances on the diagonal and the covariances off-diagonal.



The covariance matrix defines the shape of the data. Diagonal spread is captured by the covariance, while axis-aligned spread is captured by the variance.

The built-in **cov** function in MATLAB computes covariance. We can find the covariance for matrix a by:

```
cov(a) % Compute the covariance matrix for "a"
c = a-repmat(mean(a),n,1); % Subtract the mean from "a"
c'*c/(n-1) % Compute the covariance matrix for "a"
```

The covariance of the correlated noise shoud have large off-diagonal terms.

Principal Components

Principal components analysis is a dimensional reduction technique that relies on coordinate transformation. Your original data is plotted on an X-axis and Y-axis. PCA aims to rotate these two axes such that the new axis X' is along the maximum variance of the data. Your choice of X' will determine Y', as these two axes are perpendicular in PCA. So how do we get X'?

We use the ***eigenvalues*** and ***eigenvectors*** of the covariance matrix we previously calculated. These are values used to describe the behavior of a linear system of equations. In PCA, each ***eigenvector*** is a unit vector pointing in the direction of a new coordinate axis. To find X', we take the axis with the highest ***eigenvalue.***

```
sigma = cov(b) % Compute covariance matrix of b
[V, D] = eig(sigma) % V = eigenvectors, D = eigenvalues for covariance matrix
    sigma
```

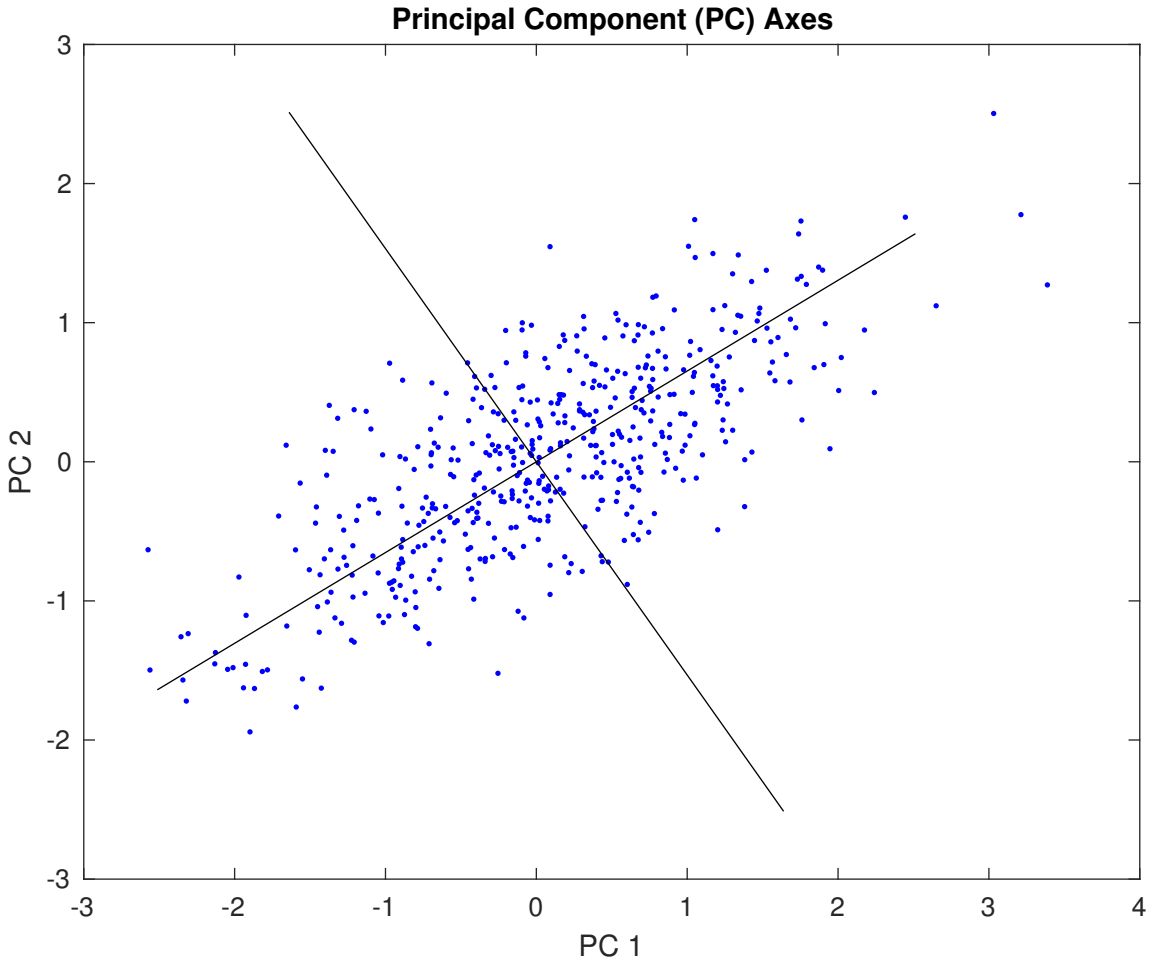
Since the noise was generated randomly, we will each have slightly different values for V and D. Here is a sample output:

$$V = \begin{bmatrix} 0.5387 & -0.8425 \\ -0.8425 & -0.5387 \end{bmatrix}$$

$$D = \begin{bmatrix} 0.2048 & 0 \\ 0 & 1.3341 \end{bmatrix}$$

The eigenvalues are stored on the diagonal of D, while the corresponding eigenvectors are the rows stored in V. The second eigenvalue (1.3341) is the largest, which means the second eigenvector [-0.8425 -0.5387] is the ***first principal component.*** This means that a vector pointing from the origin to (-0.8425, -0.5387) lies along the axis of maximum variance in the data. To plot the principal components:

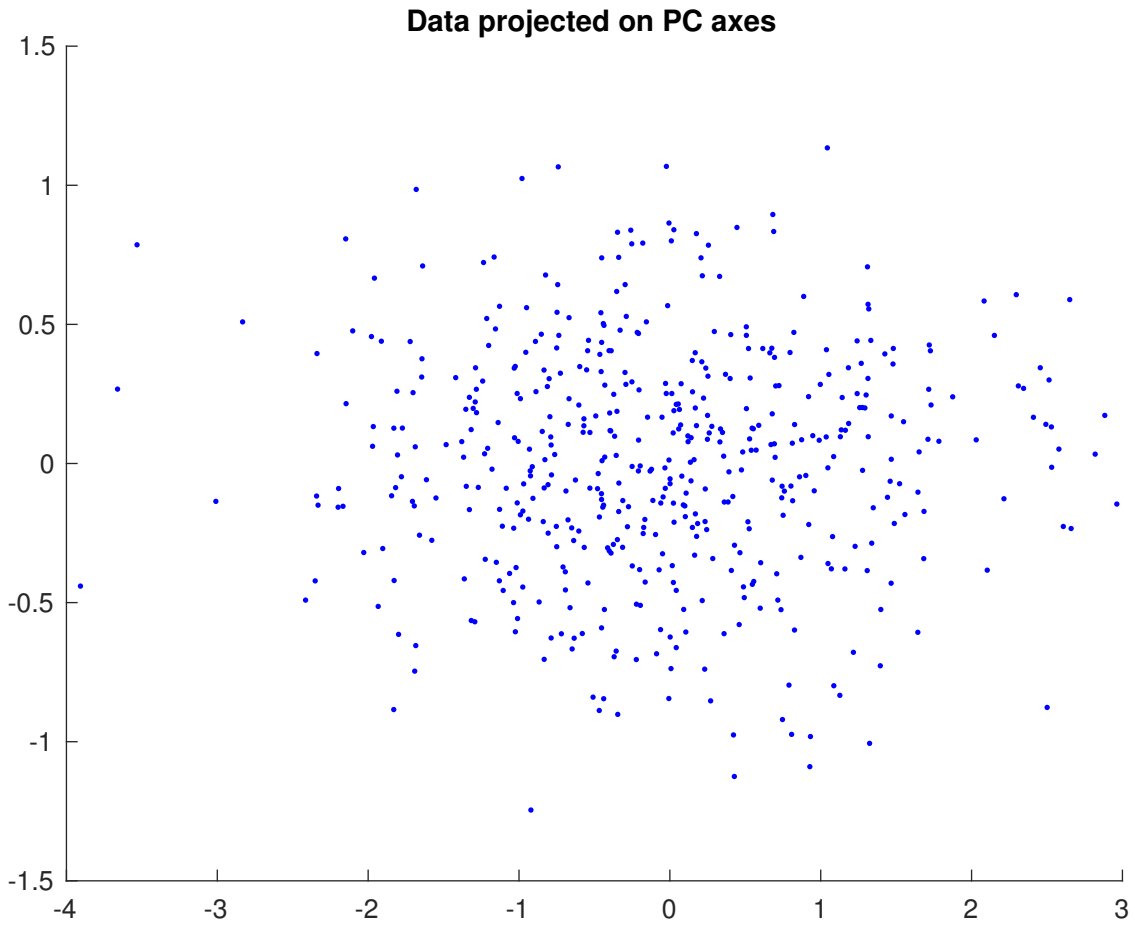
```
plot(b(:,1),b(:,2),'b.'); % Plot correlated noise
hold on
plot(3*[-V(1,1) V(1,1)], 3*[-V(1,2) V(1,2)], 'k'); % Plot axis in direction of 1
    st eigenvector
plot(3*[-V(2,1) V(2,1)], 3*[-V(2,2) V(2,2)], 'k'); % Plot axis in direction of 2
    nd eigenvector
xlabel('PC 1');
ylabel('PC 2');
title('Principal Component (PC) Axes');
```



Now you use these new coordinate axes to reassign the (X, Y) values to all your datapoints. First, you want to reorder the eigenvectors so that the first principal component is in the first row. Then, you can multiply the data by this reordered matrix to obtain the new, transformed data. For example:

```
V2(:,1) = V(:,2); % Place the 1st principal component in the 1st row
V2(:,2) = V(:,1); % Place the 2nd principal component in the 2nd row
newB = b*V2; % Project data on PC coordinates
scatter(newB(:,1), newB(:,2), 'b.');
title('Data projected on the PC axes');
```

By plotting this, we can see that we simply rotated the data so that the greatest variation likes along the X-axis.



This alone is not enough to simplify our data. To reduce it further, we can remove the data plotted on the Y-axis. The goal of PCA is that if you force as much of the variation as possible into a few dimensions, you can throw away the rest without losing much information.

How much variation can you capture by doing this? This is calculated by taking the ratio of the eigenvalue to the sum of all the eigenvalues. For example, imagine the first principal component has an eigenvalue of around 1.33 and the second principal component has an eigenvalue of around 0.20. If you keep only the first principal component, you keep 87% of the data ($0.87 = 1.33/[1.33+0.20]$). In this way, you compress the size of the data by 50% but lose only 13% of the variance.

MATLAB has a built-in function to perform these calculations called **princomp.**

```
[coeff, score, latent] = princomp(b); % Compute principal components of data in b
```

The eigenvectors are stored in the variable **coeff,** the eigenvalues are stored in **latent,** and the transformed data (old data projected onto the new PC axes) are stored in **score.** MATLAB also sorts the eigenvectors from highest to lowest.