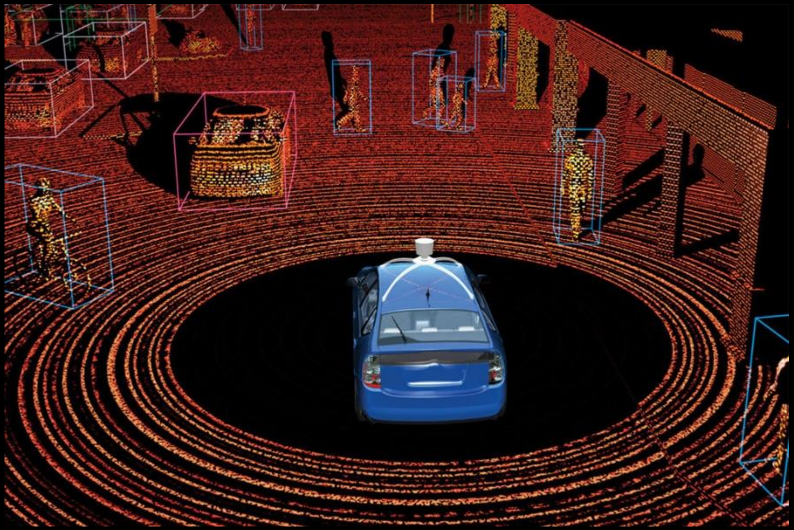
Singular Value Decomposition (SVD)
and Principle Component Analysis (PCA)

So far…

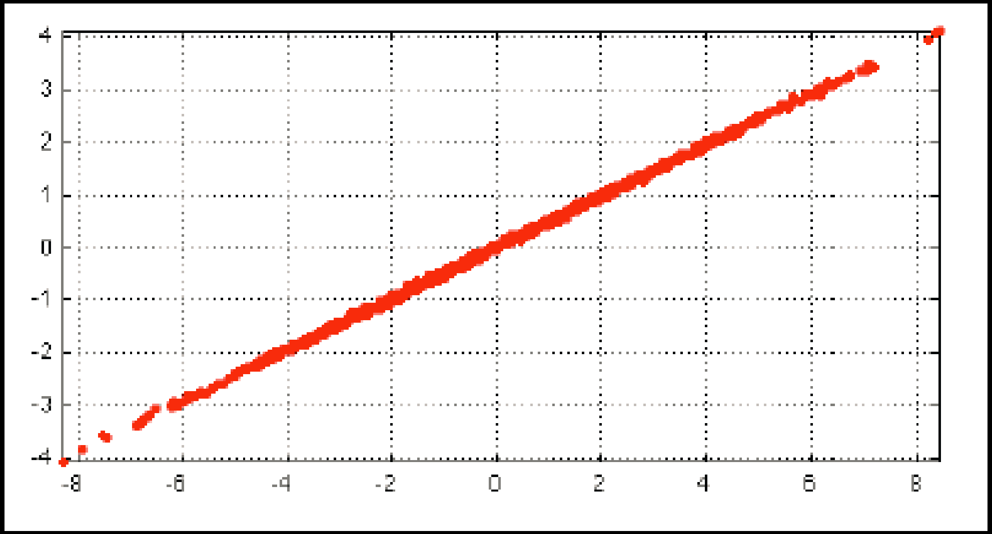
- We have been talking about planning motion
- But your planner is only as good as your perception



- Perception requires processing sensor data; i.e. finding structure in the data
- Today we'll cover some of the most basic processing you can do:
    - De-noising
    - Dimensionality reduction

Example: Dimensionality Reduction

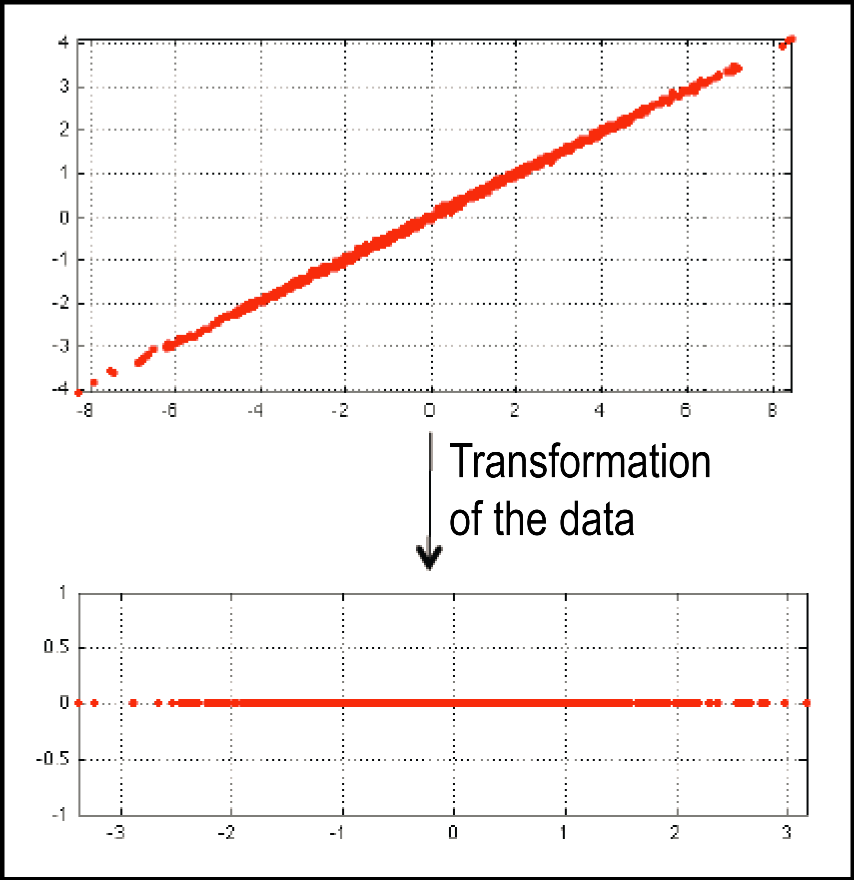
- Find a line in a 2D point cloud



- The space is 2-dimensional, but the data is 1-dimensional (a line)

How do we transform the data to get rid “unimportant” dimensions/rotations?

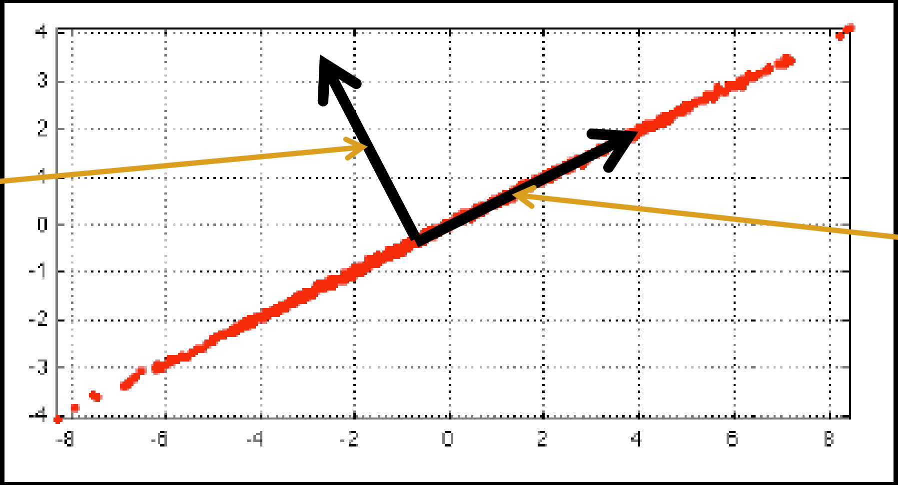
- We will see how to do this with a linear transform
- The technique is called Principle Component Analysis (PCA)
- A fundamental method in
  - Machine learning
  - Robotics
  - Computer vision
- Very important for dealing with high-dimensional data



The main idea of PCA

- We care about the variance of the data
- High-variance implies high importance
  - Why does this make sense?

Data does not vary much along this axis



Data varies a lot along this axis

- But how do we compute the high-variance axes?

Outline

- Preliminaries
  - Variance and Co-variance
  - Eigenvalues and eigenvectors
  - Singular Value Decomposition (SVD)
- Principle Component Analysis (PCA)
- Applications of PCA

Covariance

Variance

- Variance is a measure of the spread of n datapoints in some dataset X
- Let $\bar{x}$ be the mean of the datapoints
- The variance of the data is:

$$\mathrm{Var}(x) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})\,(x_i - \bar{x})}{(n-1)}$$

- Note: this is for 1-dimensional data

Covariance

- Variance is a measure of the deviation from the mean for points *in one dimension*

- Covariance is a measure of how much each of the dimensions vary from the mean *with respect to each other*

- Covariance is measured between every pair of dimensions to see the correlation between those dimensions
  - Example: time spent on homework vs. grade on homework

- The covariance between one dimension and itself is the variance

Covariance

$$\mathrm{Var}(X) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})\,(x_i - \bar{x})}{(n-1)}$$

$$\mathrm{Cov}(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})\,(y_i - \bar{y})}{(n-1)}$$

- So, if you had a 3-dimensional data set (x,y,z), then you could measure the covariance between the x and y dimensions, the y and z dimensions, and the x and z dimensions
- Measuring the covariance between x and x , or y and y , or z and z would give you the variance of the x , y and z dimensions, respectively

Covariance example

- Let's say we have a dataset consisting of two variables:
  - x_1 : Number of hours spent on homework
  - x_2 : Grade on the homework
- You find that $Cov(x_1, x_2) = 150.6$
- What does this mean?

Covariance

- Magnitude of covariance is not as important as its sign
- A positive value of covariance indicates *both dimensions increase or decrease together*
  - Example: as the number of hours spent on homework increases, the grade on that homework increases
- A negative value indicates *while one increases the other decreases*
  - Example: time spent playing video games vs time spent on homework
- If covariance is zero: the two dimensions are independent of each other
  - Example: eye color vs. grade on homework

Covariance Matrix

- Represent Covariance between dimensions as a matrix, e.g. for 3 dimensions:

$$Q = \begin{pmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{pmatrix}$$

- The diagonal is the variances of x, y and z
- cov(x,y) = cov(y,x), hence matrix is symmetric about the diagonal
- n-dimensional data will result in n x n covariance matrix

Estimating the Covariance Matrix

- First, form a dataset matrix X, with m rows (for m-dimensional data) and n columns (for the n data points)

$$X = \begin{array}{cl} \text{n datapoints} & \\ \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots \\ x_{1,2} & x_{2,2} & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix} & \text{m dimensions} \end{array}$$

- Then, estimate Q by
    1. Subtract the mean of the datapoints from every column of X
    2. $Q = \frac{XX^T}{(n-1)}$
- This is the estimator to use when you don't know what kind of distribution the data came from
- If you know the distribution, you can use distribution-specific estimators (not covered here)

Eigenvalues and eigenvectors

The eigenvalue problem

- The eigenvalue problem is any problem having the following form:

$$\mathbf{A}v = \lambda v$$

$\mathbf{A}$: n x n matrix

v: n x 1 non-zero vector

λ: scalar

- A value of λ for which this equation has a solution is called a eigenvalue of $\mathbf{A}$
- A v which corresponds to this value of λ is called an eigenvector of $\mathbf{A}$.

The eigenvalue problem

$$\begin{pmatrix} 2 & 3 \\ 2 & 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 8 \end{pmatrix} = 4 \times \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

$$\mathbf{A} \quad * \quad v \quad = \lambda * v$$

Therefore, $v = (3,2)$ is an *eigenvector* of the matrix $\mathbf{A}$ and $\lambda = 4$ is an *eigenvalue* of $\mathbf{A}$

We will see how to calculate eigenvalues/eigenvectors later

Properties of eigenvectors and eigenvalues

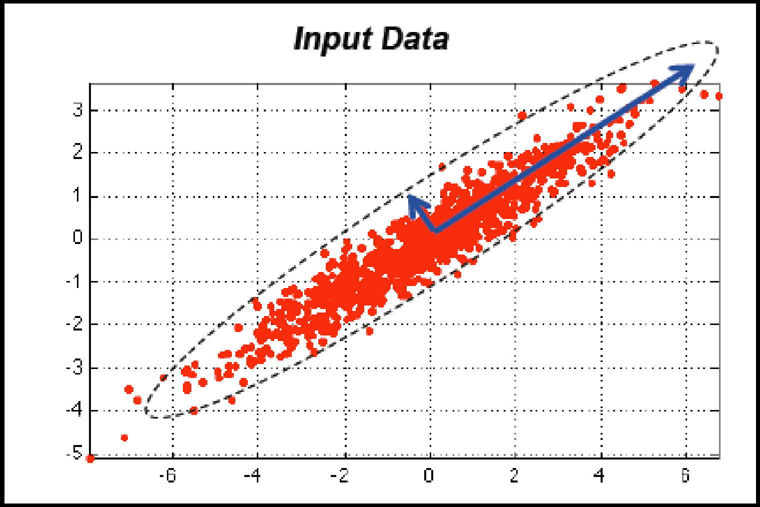
- Note that irrespective of how much we scale (3,2), the solution is always a multiple of 4.
- Eigenvectors can *only be found for square matrices* and not every square matrix has eigenvectors.
- Given an n x n matrix, we can find n eigenvectors

Properties of eigenvectors and eigenvalues

- All eigenvectors of a matrix are *orthogonal* to each other
- In practice eigenvectors are normalized to have unit length
  - Since the length of the eigenvectors do not affect our calculations we prefer to keep them standard by scaling them to have a length of 1

Eigenvectors of covariance matrix

- Eigenvectors of Q with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset
- Eigenvectors of the covariance matrix are called principle components



- But how do we compute eigenvalues and eigenvectors?
  - There are many ways
  - We will see how to do it using SVD

Singular Value Decomposition (SVD)

SVD

- SVD decomposes any matrix M into the following form:

$$M = U * \Sigma * V^T$$

$$m \times n = m \times m \quad m \times n \quad n \times n$$

Columns of U are eigenvectors of MM^T

SVD

- SVD decomposes any matrix M into the following form:

$$M = U * \Sigma * V^T$$

$$m \times n = m \times m \quad m \times n \quad n \times n$$

$$\Sigma \equiv \begin{bmatrix} \sigma_{\tilde{1}} & & & & \\ & \ddots & & & 0 \\ & & \sigma_{\tilde{r}} & & \\ & & & 0 & \\ & 0 & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

σs are $\sqrt{\text{eigenvalues}}$ of M^TM and MM^T
in decreasing order of magnitude

SVD

- SVD decomposes any matrix M into the following form:

$$M \quad = \quad U \quad * \quad \Sigma \quad * \quad V^T$$

$$m \times n \quad = \quad m \times m \qquad m \times n \qquad n \times n$$

Columns of V are eigenvectors of M^TM

SVD

- SVD decomposes any matrix M into the following form:

$$M = U * \Sigma * V^T$$

$$m \times n = m \times m \quad m \times n \quad n \times n$$

- Columns of U and V are orthonormal:
  - Each column vector has unit magnitude
  - Each column vector is orthogonal to the others

SVD

- SVD decomposes any matrix M into the following form:

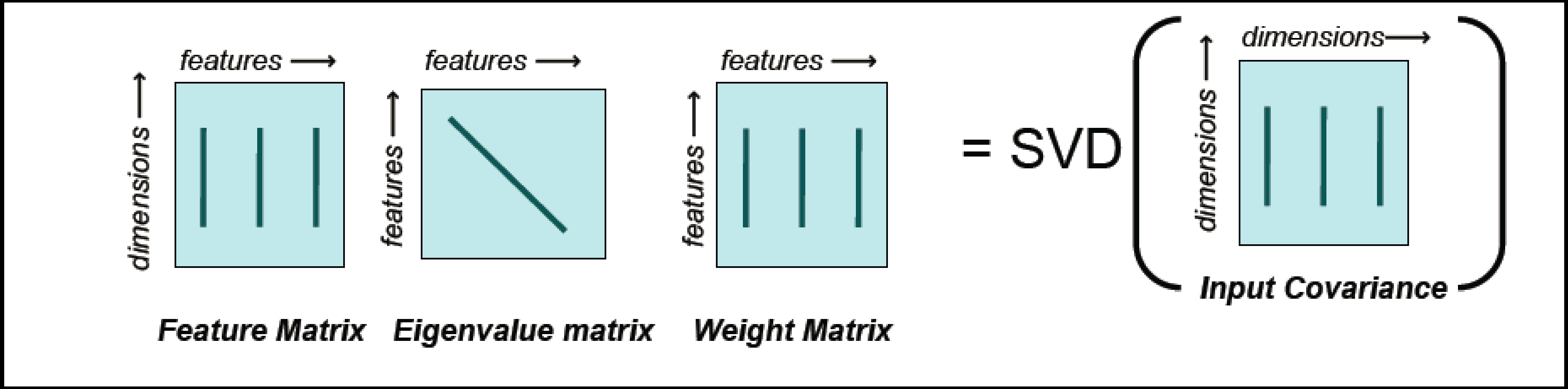
$$M = U * \Sigma * V^T$$

$$m \times n = m \times m \quad m \times n \quad n \times n$$

- Computing $[U, \Sigma, V^T] = SVD(M)$ is complicated
  - Many tools available, e.g. in Eigen library (C++) or Matlab or numpy
  - Complexity is $O(4m^2n + 8mn^2 + 9n^3)$

Back to Eigenvalues and Eigenvectors

- Remember, we want to compute eigenvectors and eigenvalues of Q (the covarance matrix)
- $[U, \Sigma, V^T]$ = SVD(Q)
- Σ contains $\sqrt{\text{eigenvalues}}$ of Q
- V contains eigenvectors of Q



Principle Component Analysis (PCA)

PCA Motivation

- **Principal Components Analysis (PCA)** is a technique that can be used to
  - Remove rotation in a dataset
  - Reduce the dimensionality of a dataset

- PCA computes a linear transformation that chooses a new coordinate system for the data set such that
  - The greatest variance by any projection of the data set comes to lie on the first axis (called the first principal component)
  - the second greatest variance on the second axis (2^{nd} principle component)
  - and so on…

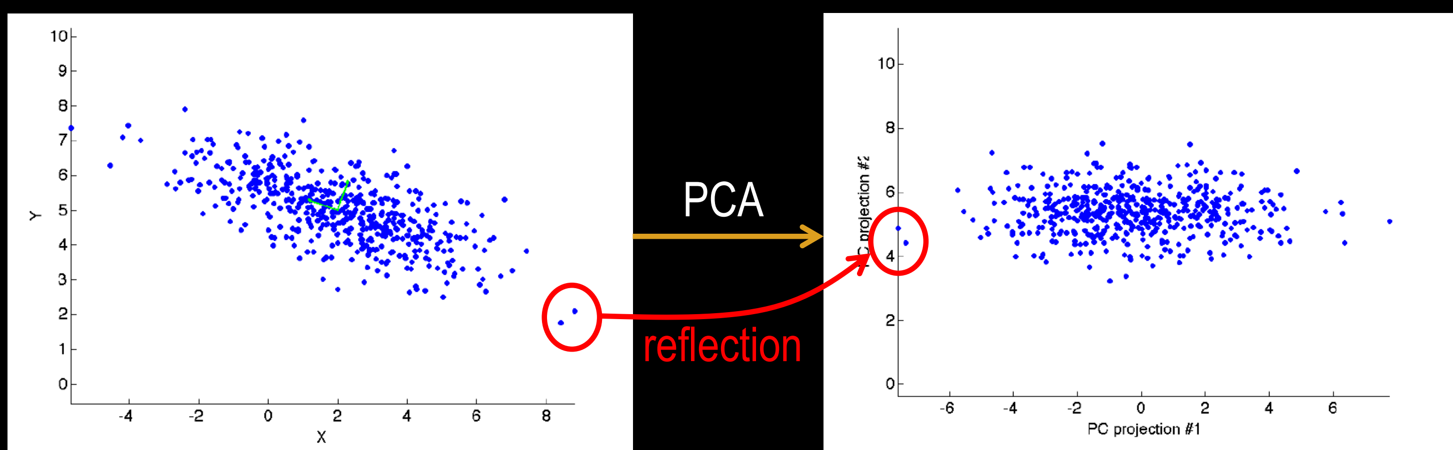
PCA algorithm to remove rotation (preserves dimension)

1. Given a dataset X
2. Compute the mean of X, μ
3. $X = X - \mu$ (subtract μ from every point in X)
4. Compute the covariance of X, $Q = \frac{XX^T}{(n-1)}$
5. SVD(Q) = $U\Sigma V^T$
   - Each column of V is a principle component
6. Compute $X_{new} = V^T X$

(Note: this may cause reflection)

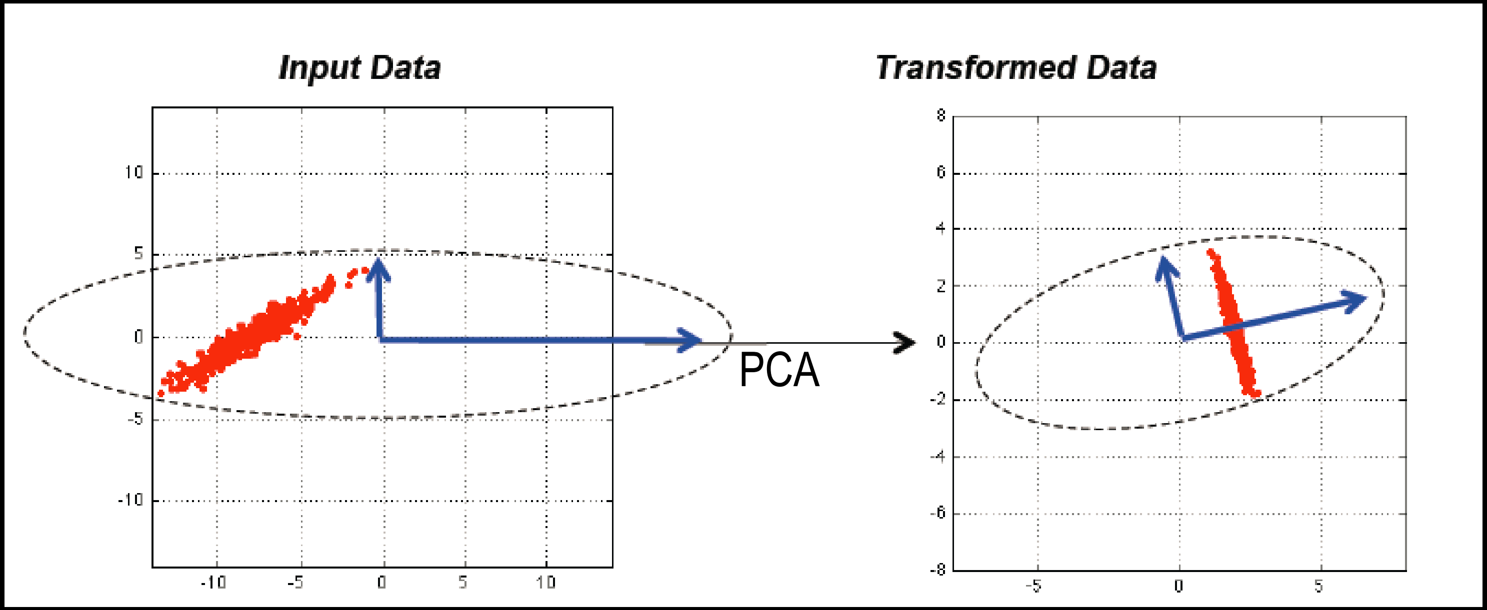
PCA Intuition for Rotation

- PCA rotates (and maybe reflects) X about the mean to align with the principal components
- PCA moves as much of the variance as possible (using an orthogonal transformation) into the first few dimensions



Why subtract the mean?

- If we don't subtract the mean, we get undesirable results:



- A mean of 0 is needed for finding a basis that minimizes the mean square error of the approximation of the data. [Miranda et al., *Neural Processing Letters*, 2008]

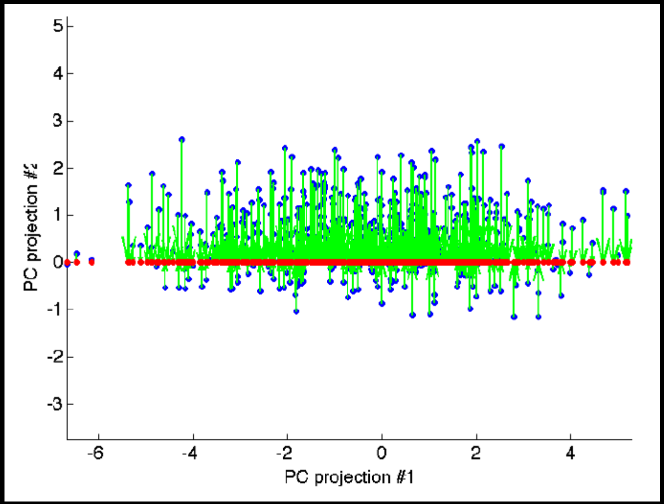
PCA algorithm to remove rotation *and reduce dimension*

1. Given a dataset X
2. Compute the mean of X, μ
3. $X = X - \mu$ (subtract μ from every point in X)
4. Compute the covariance of X, $Q = \frac{XX^T}{(n-1)}$
5. SVD(Q) $= U\Sigma V^T$
   - Each column of V is a principle component
6. Compute the variance of each principle component: $s = \mathrm{diag}(\Sigma)^2$
7. Remove all columns of V whose corresponding entry in s is less than some small threshold, call this new matrix V_s
8. Compute $X_{new} = V_s^T X$

(Note: this may cause reflection)

PCA Intuition for Dimensionality Reduction

- After rotation to align with principle components, variance in the remaining dimensions tend to be small and can be dropped with minimal loss of information
- PCA computes the optimal orthogonal transformation for keeping the subspace that has largest variance



PCA reducing dimension from 2 to 1

Do we need the covariance?

- Computing covariance can be expensive for huge datasets
- It turns out the principle components of the following matrices are the same:

$$Q = \frac{XX^T}{(n-1)}$$

$$Y = \frac{X^T}{\sqrt{(n-1)}}$$

- So, you don’t need to compute the covariance!

PCA algorithm without covariance

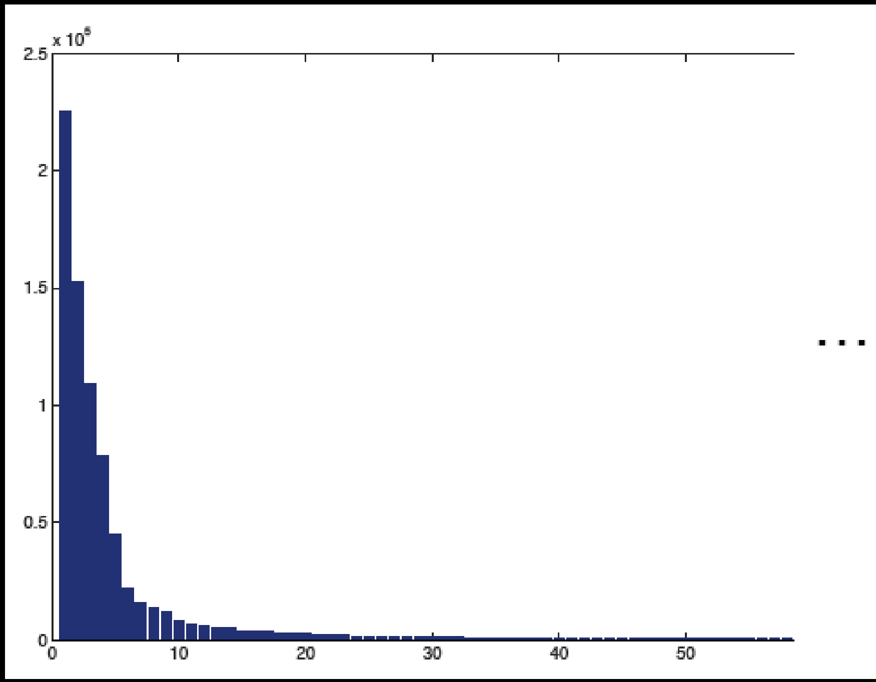
1. Given a dataset X
2. Compute the mean of X, μ
3. $X = X - \mu$ (subtract μ from every point in X)
4. Compute $Y = \frac{X^T}{\sqrt{(n-1)}}$ ← Before: Compute the covariance of X, Q = X* X^T / (N-1)
5. SVD(Y) $= U\Sigma V^T$
   - Each column of V is a principle component
6. Compute the variance of each principle component: $s = \mathrm{diag}(\Sigma)^2$
7. Remove all columns of V whose corresponding entry in s is less than some small threshold, call this new matrix V_s
8. Compute $X_{new} = V_s^T X$

(Note: this may cause reflection)

How many dimension do we want?

- No general answer
  - Depends on application
- Often, high-dimensional data can be described with much fewer variables
- Diagonal of Σ tells us which eigenvectors are important

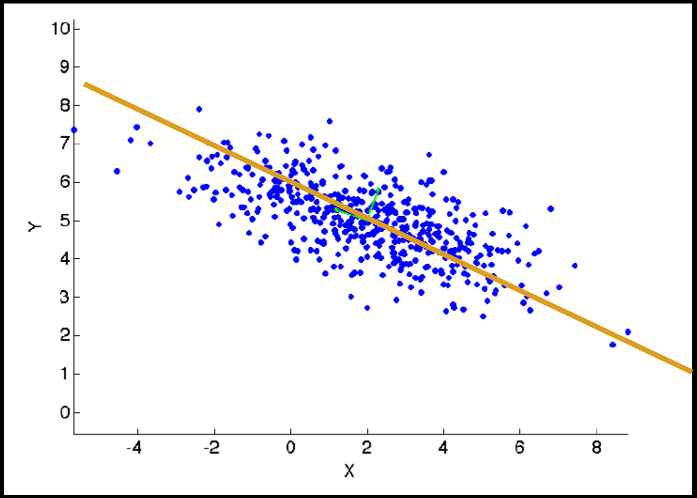
$$\mathbf{\Sigma} \equiv \begin{bmatrix} \sigma_{\tilde{1}} & & & & \\ & \ddots & & & 0 \\ & & \sigma_{\tilde{r}} & & \\ & & & 0 & \\ & 0 & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$



- Eigenvalues of 1200 dimensional video data
- Not much variance after component 30, so we can throw out the remaining 1170 components!

PCA relation to least-squares regression

- The first principal component corresponds to a line that passes through mean and minimizes the sum of squares of the distances of the points from the line
- The second principal component corresponds to the same concept *after all correlation with the first principal component has been subtracted from the points.*
- Each eigenvalue is proportional to the portion of the sum of the squared distances of the points from their mean that is correlated with each eigenvector
- The sum of all the eigenvalues is equal to the sum of the squared distances of the points from their mean



Limitations of PCA: Variable scaling

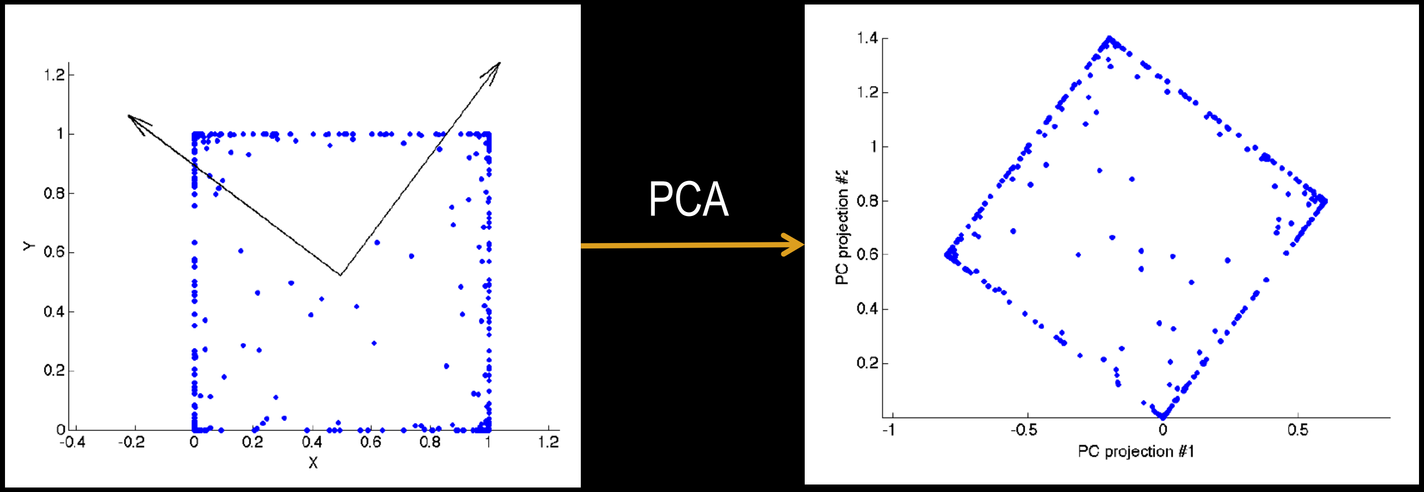
- PCA is sensitive to the scaling of the variables
- Example:
  - Say x_1 and x_2 have the same variance and positive correlation

    45º rotation
  - PCA will produce a rotation of 45° so that the two variables will be equally distributed around the 1st principle component
  - But, if we scale all the values of x_1 by 1000, what will the 1st P.C. produced by PCA be?

    1º rotation
    - 1st P.C. will be almost exactly equal to x_1
    - 2nd P.C will be aligned with x_2
- WARNING: When the variables have different units (e.g. height and temperature), PCA doesn't make much sense
  - E.g. Would get different results for Fahrenheit vs. Celsius

Limitations of PCA: Non-linear manifolds

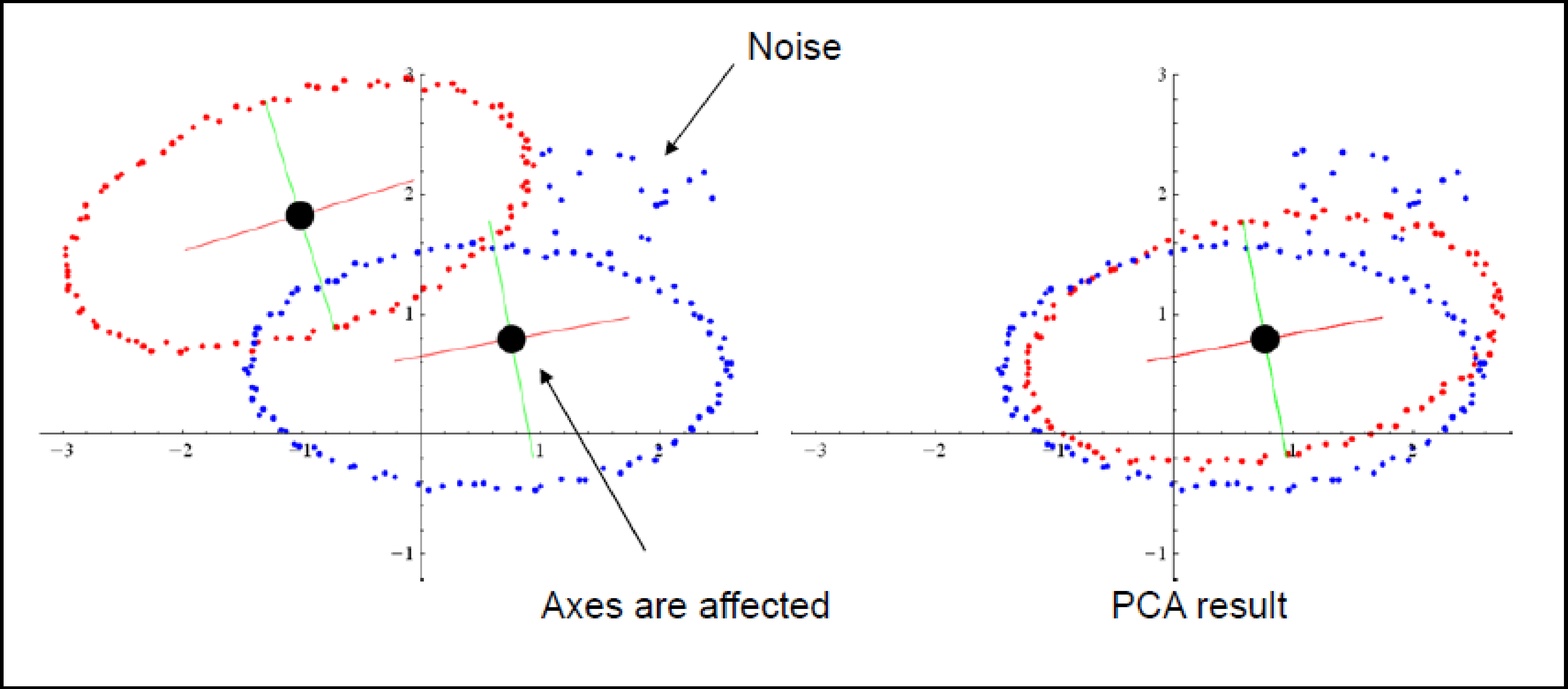
- PCA can only produce linear transforms
- What if the manifold of data is not linear?



- PCA performs poorly
- Nonlinear dimensionality reduction techniques exist
  - They are much more computationally-expensive ☹

Limitations of PCA: Sensitivity to Noise

- PCA used to align the axes of two datasets:



Example from Tao Ju

Break

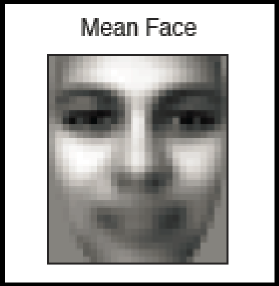
Applications of PCA

Application: PCA on Image data

- Eigenfaces: Analysis of a database of face images
  - Question: What are good features to represent faces?
- Each image is grayscale, has 780 pixels
  - A sample is just these pixel values lined up in a vector

Eigenfaces

- The top principle components:



Principle components

How well does this work?

- Advantage: Instead of using 780 pixel values, can represent a face as a weighted combination of eigenfaces
- Test:

Input face

Approximation of input face with weighted combination of eigenfaces

PCA for video data

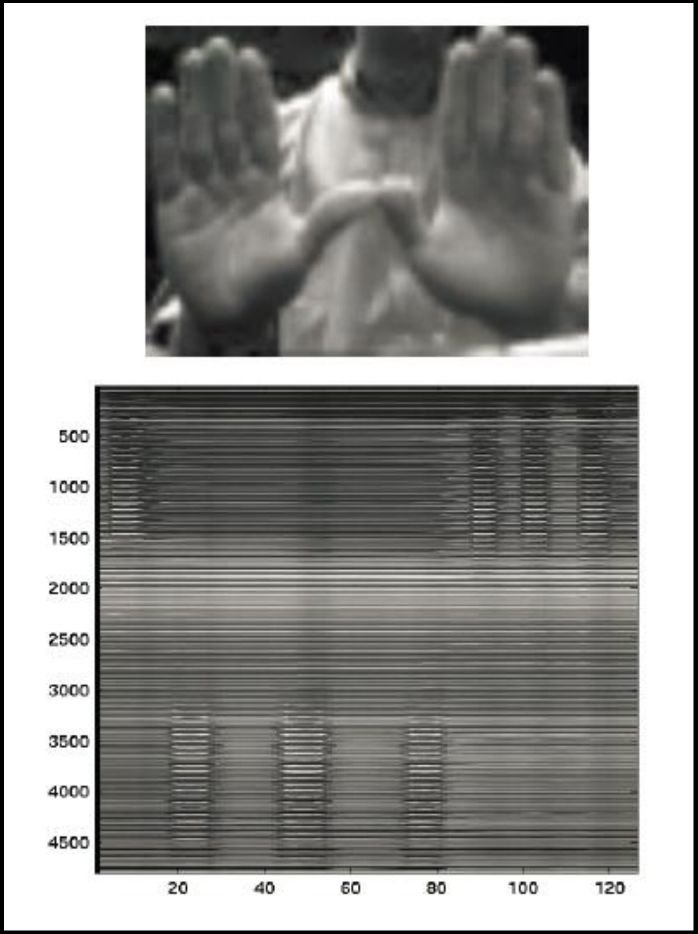
- Problem: Sometimes the data is too high-dimensional
  - Example: videos 1280x720xT = 921,600D x T frames
- SVD will take a LONG time to run!
- There are many variant and extensions of PCA
- Useful approach for video is the Expectation Maximization PCA (EM-PCA)
  - Main idea: Alternate between successive approximations
    - Start with random matrix **C** and loop over:
      - $Z = C^{+}X$
      - $C = XZ^{+}$
    - After convergence **C** spans the PCA space
  - If we choose a low rank **C** then computations are significantly more efficient than the SVD

PCA for online data

- Problem: Sometimes we have too many data samples
  - Irrespective of the dimensionality
  - Example: long video recordings
- Incremental SVD algorithms
  - Update the $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ matrices with only a small subset or a single sample point
  - Very efficient updates

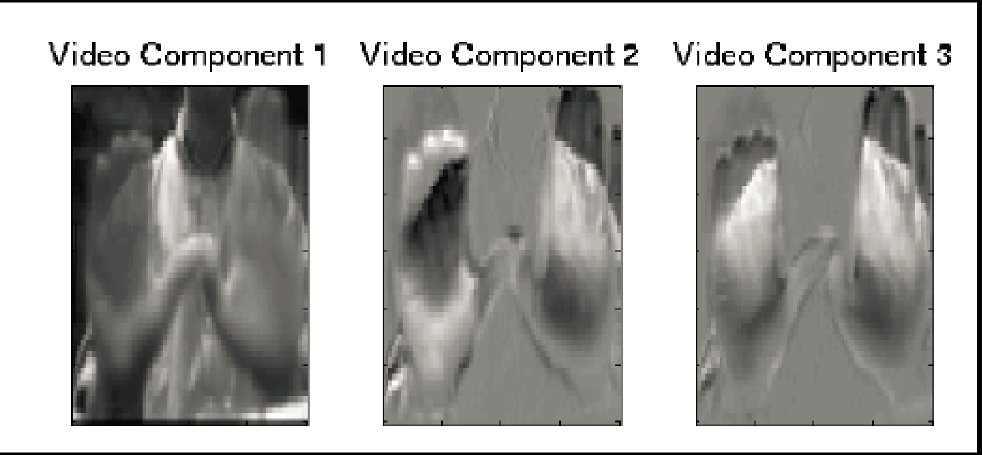
Example: PCA for video

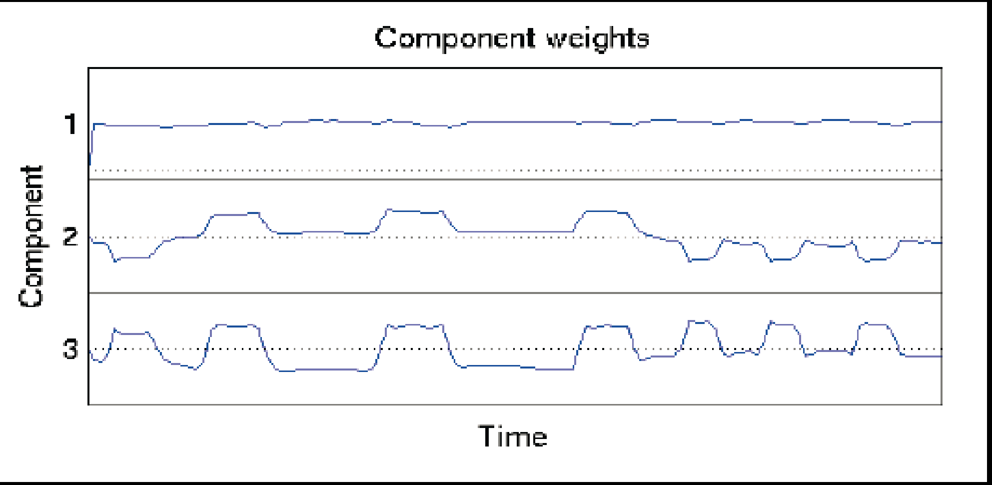
- A movie is a series of frames
  - Each frame is a data point
  - 126 frames, 80x60 pixels per frame
  - Data will be 4800x126



PCA for video results

- Just like with eigenfaces, each image can be represented as a weighted combination of principle components
- Advantage: only need to store the principle components and weights for each frame
- We just learned a method for video compression!





Summary

- The main assumption: variance of the data correlates with what matters
  - The higher the variance, the more important
- PCA aligns the dataset so that most variance is captured by first few components
- PCA can be used for
  - Rotating data to more reasonable axes
  - Reducing the dimension of the data
- PCA uses SVD to find the eigenvalues of the covariance matrix (or the Y matrix)
- Many high-dimensional datasets have hidden low-dimensional structure
  - Can often represent high-dimensional with very few variables
    - E.g. eigenfaces, video
- Limitations of PCA: Can only do linear transforms, all the data should be in the same units

Homework

- Read Point set registration (up to and including ICP)