Unsupervised Learning: Dimension Reduction

by Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents

1. Principal Component Analysis (PCA)

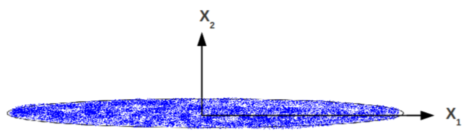
Motivation: Can we describe high-dimensional data in a "simpler" way?

$\longrightarrow$ Dimension reduction without losing too much information
$\longrightarrow$ Find a low-dimensional, yet useful representation of the data

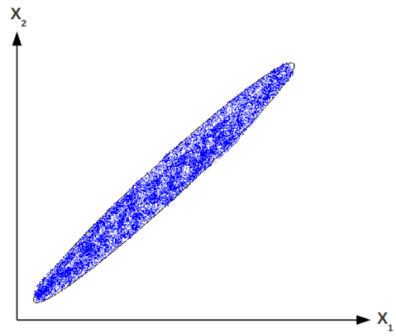
- Why dimensionality reduction?
  - insights into the low-dimensinal structures in the data (visualization)
  - Fewer dimensions ⇒ Less chances of overfitting ⇒ Better generalization
  - Speeding up learning algorithms
    - Most algorithms scale badly with increasing data dimensionality
  - Less storage requirements (data compression)
  - Note: Dimensionality Reduction is different from Feature Selection
    - .. although the goals are kind of the same
  - Dimensionality reduction is more like “Feature Extraction”
    - Constructing a small set of new features from the original features

- How?
  idea: highly correlated data contains redundant features

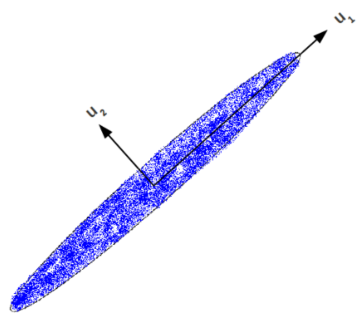
- Each example x has 2 features $\{x_1, x_2\}$
- Consider ignoring the feature x_2 for each example
- Each 2-dimensional example x now becomes 1-dimensional $x = \{x_1\}$
- Are we losing much information by throwing away x_2 ?
- No. Most of the data spread is along x_1 (very little variance along x_2)



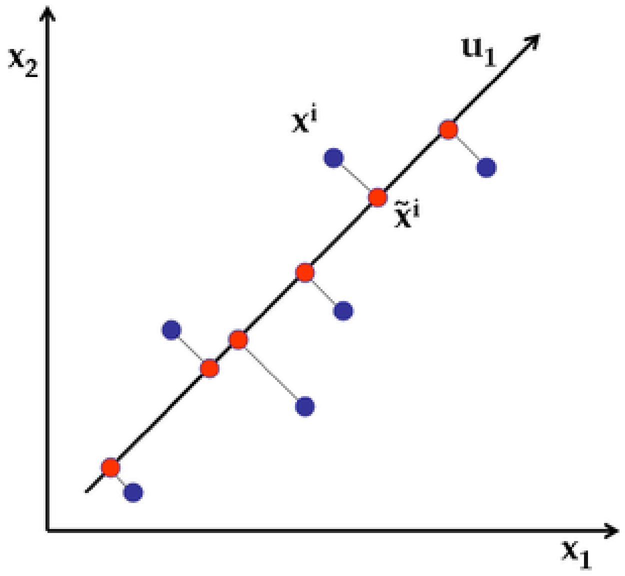
- Each example x has 2 features $\{x_1, x_2\}$
- Consider ignoring the feature x_2 for each example
- Each 2-dimensional example x now becomes 1-dimensional $x = \{x_1\}$
- Are we losing much information by throwing away x_2 ?
- Yes, the data has substantial variance along both features (_i.e._ both axes)

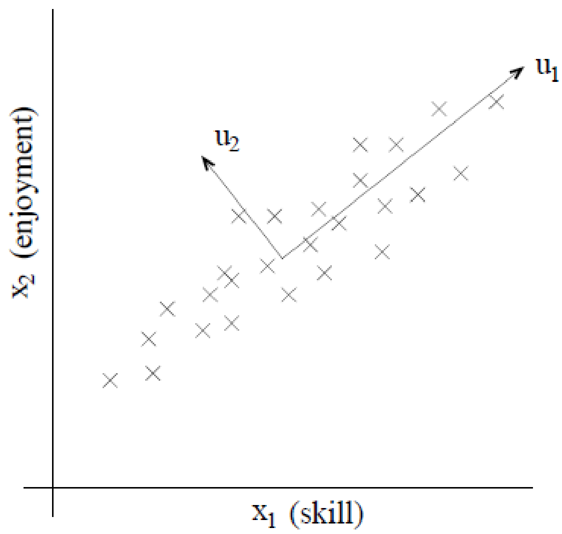


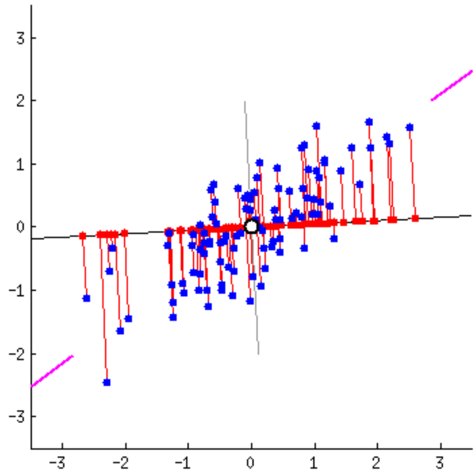
- Now consider a change of axes
- Each example x has 2 features $\{u_1, u_2\}$
- Consider ignoring the feature u_2 for each example
- Each 2-dimensional example x now becomes 1-dimensional $x = \{u_1\}$
- No. Most of the data spread is along u_1 (very little variance along u_2)



- Data $\rightarrow$ projection onto unit vector $\hat{u}_1$
  - PCA is used when we want projections capturing maximum variance directions
  - Principal Components (PC): directions of maximum variability in the data
  - Roughly speaking, PCA does a change of axes that can represent the data in a succinct manner







- HOW?
  1. Maximize variance (most separable)
  2. Minimize the sum-of-squares (minimum squared error)

2. PCA Algorithm

2.1. Pre-processing

- Given data

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}, \qquad X = \begin{bmatrix} \cdots & (x^{(1)})^T & \cdots \\ \cdots & (x^{(2)})^T & \cdots \\ & \vdots & \\ \cdots & (x^{(m)})^T & \cdots \end{bmatrix}$$

- Shifting (zero mean) and rescaling (unit variance)
  1. Shift to zero mean

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$
$$x^{(i)} \leftarrow x^{(i)} - \mu \quad \text{(zero mean)}$$

  2. [optional] Rescaling (unit variance)

$$\sigma_j^2 = \frac{1}{m-1}\sum_{i=1} m\left(x_j^{(i)}\right)^2$$
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)}}{\sigma_j}$$

2.2. Maximize variance

- Find unit vector u such that maximizes variance of projections

  Note: $m \approx m-1$ for big data

$$\begin{aligned} \text{variance of projected data} &= \frac{1}{m}\sum_{i=1}^{m}\left(u^T x^{(i)}\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)^T} u\right)^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)^T} u\right)^T \left(x^{(i)^T} u\right) = \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} x^{(i)^T} u \\ &= u^T\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}\right) u \\ &= u^T S u \qquad \left(S = \frac{1}{m} X^T X : \text{sample covariance matrix}\right) \end{aligned}$$

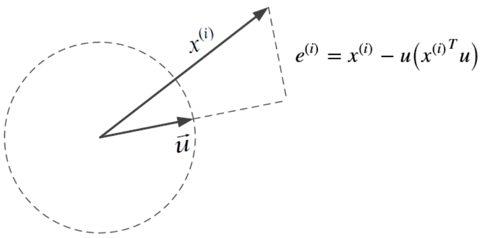
- In an optimization form

$$\begin{aligned} \text{maximize} \quad & u^T S u \\ \text{subject to} \quad & u^T u = 1 \end{aligned}$$

$$u^T S u = u^T \lambda u = \lambda u^T u = \lambda \quad (\text{Eigen analysis} : Su = \lambda u)$$

$$\begin{aligned} &\Longrightarrow \text{ pick the largest eigenvalue } \lambda_1 \text{ of covariance matrix } S \\ &\Longrightarrow u = u_1 \text{ is the } \lambda_1' s \text{ corresponding eigenvector} \\ &\Longrightarrow u_1 \text{ is the first principal component (direction of highest variance in the data)} \end{aligned}$$

2.3. Minimize the sum-of-squared error



$$\begin{aligned}\|e^{(i)}\|^2 &= \|x^{(i)}\|^2 - \left(x^{(i)^T} u\right)^2 \\ &= \|x^{(i)}\|^2 - \left(x^{(i)^T} u\right)^T \left(x^{(i)^T} u\right) \\ &= \|x^{(i)}\|^2 - u^T x^{(i)} x^{(i)^T} u\end{aligned}$$

$$\begin{aligned}\frac{1}{m}\sum_{i=1}^{m}\|e^{(i)}\|^2 &= \frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2 - \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} x^{(i)^T} u \\ &= \frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2 - u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}\right) u\end{aligned}$$

- In an optimization form

$$\text{minimize } \underbrace{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2}_{\text{constant given } x_i} - \underbrace{u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}\right) u}_{\text{maximize}}$$

$$\Longrightarrow \text{maximize } u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}\right) u = \max \; u^T S u$$

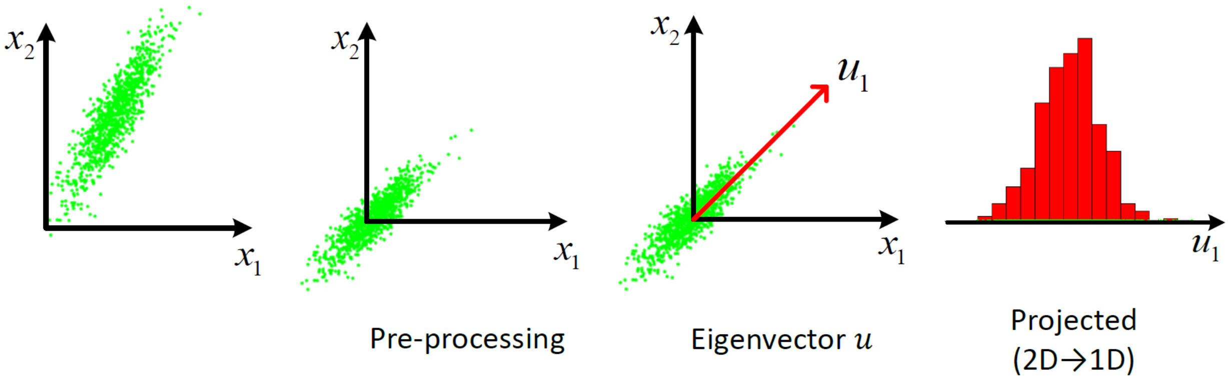
$$\therefore \; \text{minimize } \mathit{error}^2 = \text{maximize } \mathit{variance}$$

2.4. Dimension Reduction method ($n \rightarrow k$)

1. Choose top k (orthonormal) eigenvectors, $U = [u_1, u_2, \cdots, u_k]$
2. Project x_i onto span $\{u_1, u_2, \cdots, u_k\}$

$$z^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \quad \text{or} \quad z = U^T x$$

- Pictorial summary of PCA



$$x^{(i)} \rightarrow \text{projection onto unit vector } u \implies u^T x^{(i)} = \text{distance from the origin along } u$$

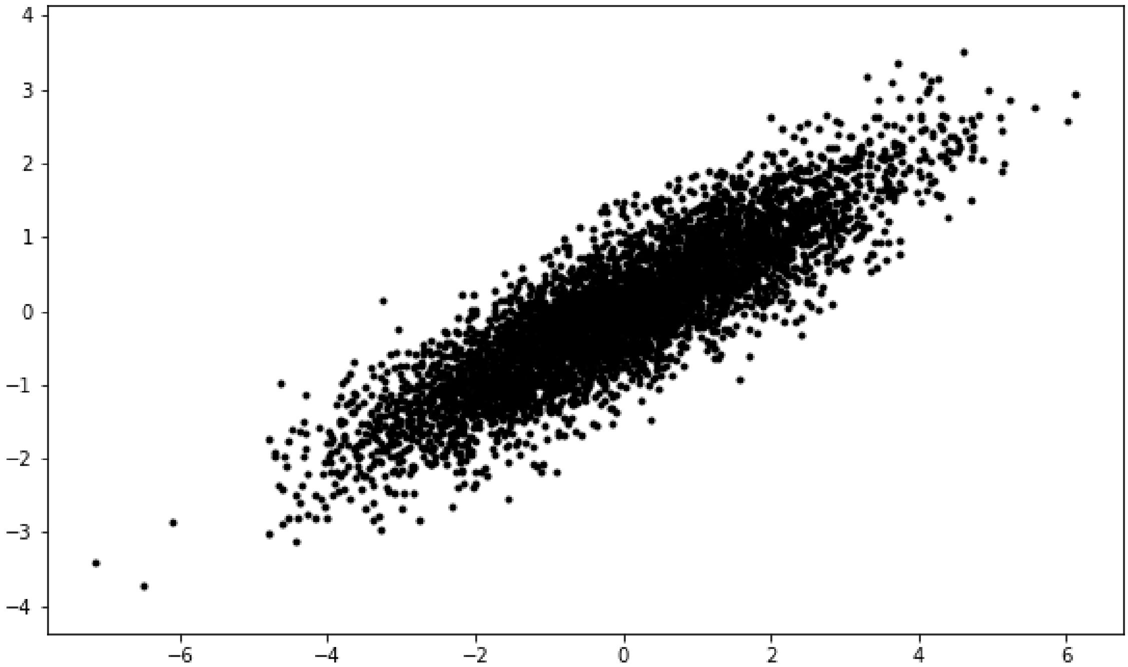
3. Python Codes

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

%matplotlib inline
```

In [2]:

```
# data generation
m = 5000
mu = np.array([0, 0])
sigma = np.array([[3, 1.5],
                  [1.5, 1]])

X = np.random.multivariate_normal(mu, sigma, m)
X = np.asmatrix(X)

fig = plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'k.')
plt.axis('equal')
plt.show()
```



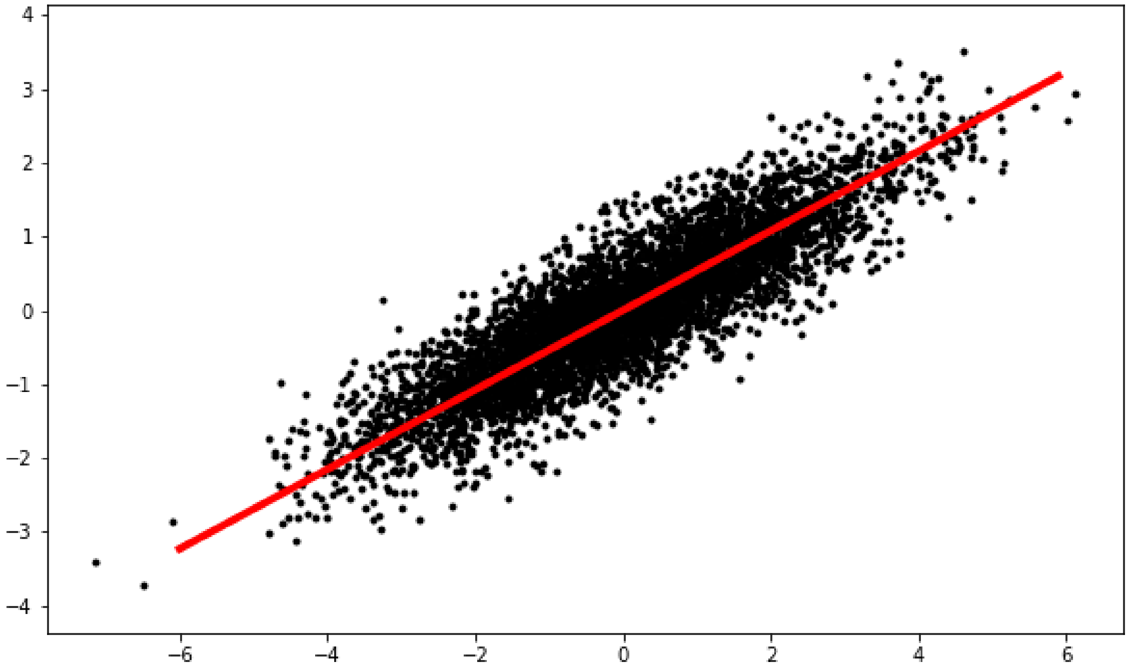
In [3]:

```
S = 1/(m-1)*X.T*X
S = np.asmatrix(S)

D, V = np.linalg.eig(S)

idx = np.argsort(-D)
D = D[idx]
V = V[:,idx]

print(D)
print(V)
```

```
[ 3.78868797  0.19209281]
[[ 0.88056479 -0.47392579]
 [ 0.47392579  0.88056479]]
```

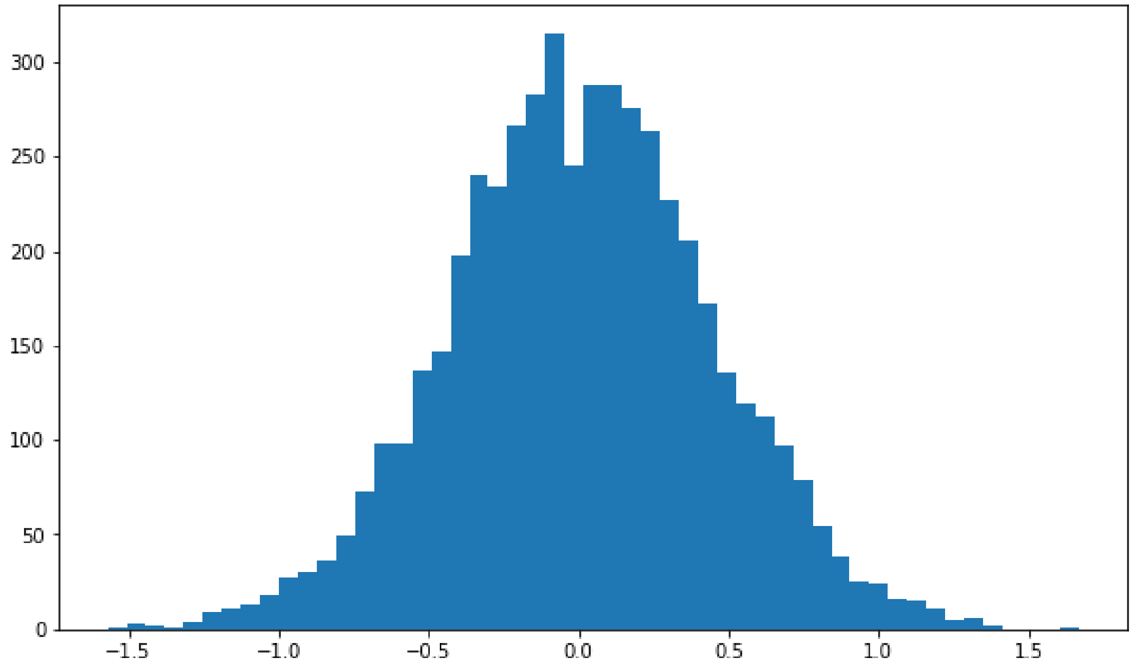
In [4]:

```
h = V[1,0]/V[0,0]
xp = np.arange(-6, 6, 0.1)
yp = h*xp

fig = plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'k.')
plt.plot(xp, yp, 'r', linewidth=4.0)
plt.axis('equal')
plt.show()
```



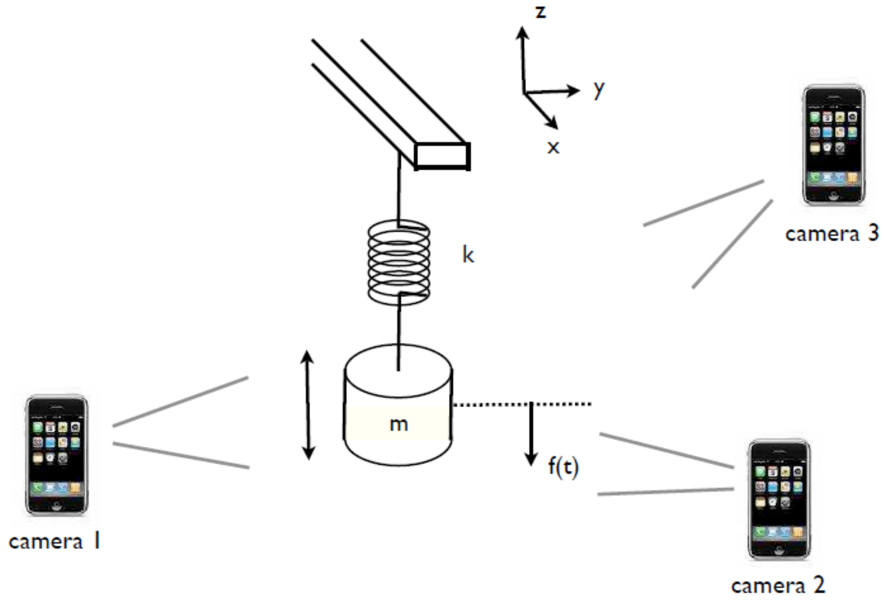
In [5]:

```
Z = X*V[:,1]

plt.figure(figsize=(10, 6))
plt.hist(Z, 51)
plt.show()
```



4. PCA Example

- multiple video camera records of spring and mass system



- source:
  - [https://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf (https://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf)](https://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf)

In [6]:

```
%%html
<center><iframe
width="560" height="315" src="https://www.youtube.com/embed/Pkit-64g0eU" frameborder
="0" allowfullscreen>
</iframe></center>
```

In [7]:

```
%%html
<center><iframe
width="560" height="315" src="https://www.youtube.com/embed/x4lvjVjUUqg" frameborder
="0" allowfullscreen>
</iframe></center>
```

In [8]:

```
%%html
<center><iframe
width="560" height="315" src="https://www.youtube.com/embed/2t62WkNIqxY" frameborder
="0" allowfullscreen>
</iframe></center>
```

$$x^{(i)} = \begin{bmatrix} x \text{ in camera 1} \\ y \text{ in camera 1} \\ x \text{ in camera 2} \\ y \text{ in camera 2} \\ x \text{ in camera 3} \\ y \text{ in camera 3} \end{bmatrix}, \qquad X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ (x^{(1)}) & (x^{(2)}) & \cdots & (x^{(m)}) \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

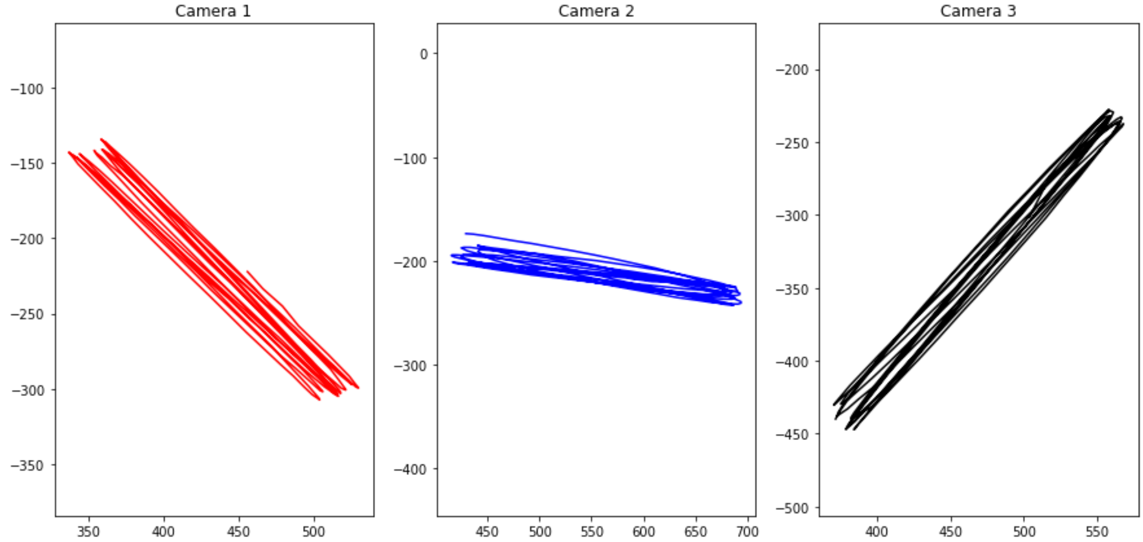
In [4]:

```
from six.moves import cPickle

X = cPickle.load(open('./data_files/pca_spring.pkl','rb'))
X = np.asmatrix(X.T)

print(X.shape)
m = X.shape[0]
```

```
(273, 6)
```

In [5]:

```
plt.figure(figsize=(15, 7))
plt.subplot(131)
plt.plot(X[:,0], -X[:,1],'r')
plt.axis('equal')
plt.title('Camera 1')

plt.subplot(132)
plt.plot(X[:,2], -X[:,3],'b')
plt.axis('equal')
plt.title('Camera 2')

plt.subplot(133)
plt.plot(X[:,4], -X[:,5],'k')
plt.axis('equal')
plt.title('Camera 3')

plt.show()
```



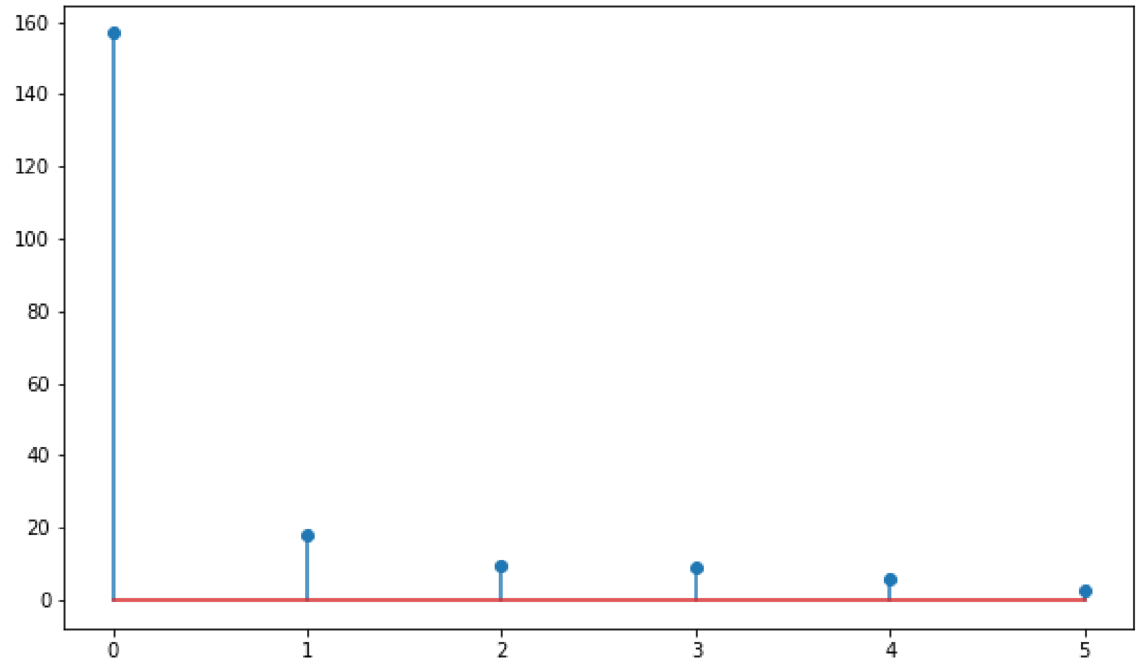
In [6]:

```
X = X - np.mean(X,axis=0)

S = 1/(m-1)*X.T*X
S = np.asmatrix(S)

D, V = np.linalg.eig(S)

idx = np.argsort(-D)
D = D[idx]
V = V[:,idx]

print(D)
print(V)
```

```
[  2.46033089e+04   3.22747042e+02   8.73851124e+01   8.19527660e+01
   3.19467195e+01   7.42861585e+00]
[[ 0.36881064  0.62298194 -0.12571821 -0.42647348  0.52121775 -0.0807439
 ]
 [ 0.35632379  0.57286174  0.132303    0.59881765 -0.40143215  0.0873404
5]
 [ 0.58419477 -0.22610057 -0.20325551 -0.47751523 -0.58153918  0.0085780
4]
 [ 0.08652315 -0.02671281  0.75692234 -0.14177391 -0.06010869 -0.6286142
2]
 [ 0.4159798  -0.29900638  0.49374948  0.05637591  0.32442517  0.6207555
9]
 [-0.46389987  0.37746931  0.32963322 -0.45633202 -0.34660023  0.4530840
3]]
```

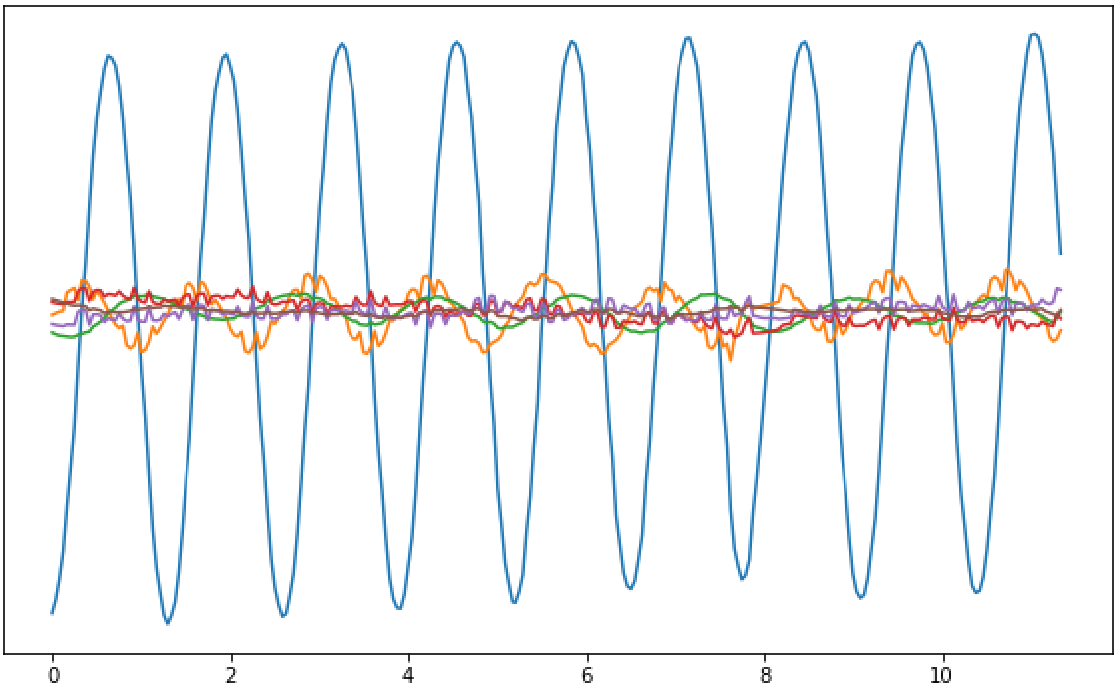
In [7]:

```
plt.figure(figsize=(10,6))
plt.stem(np.sqrt(D))
plt.show()
```



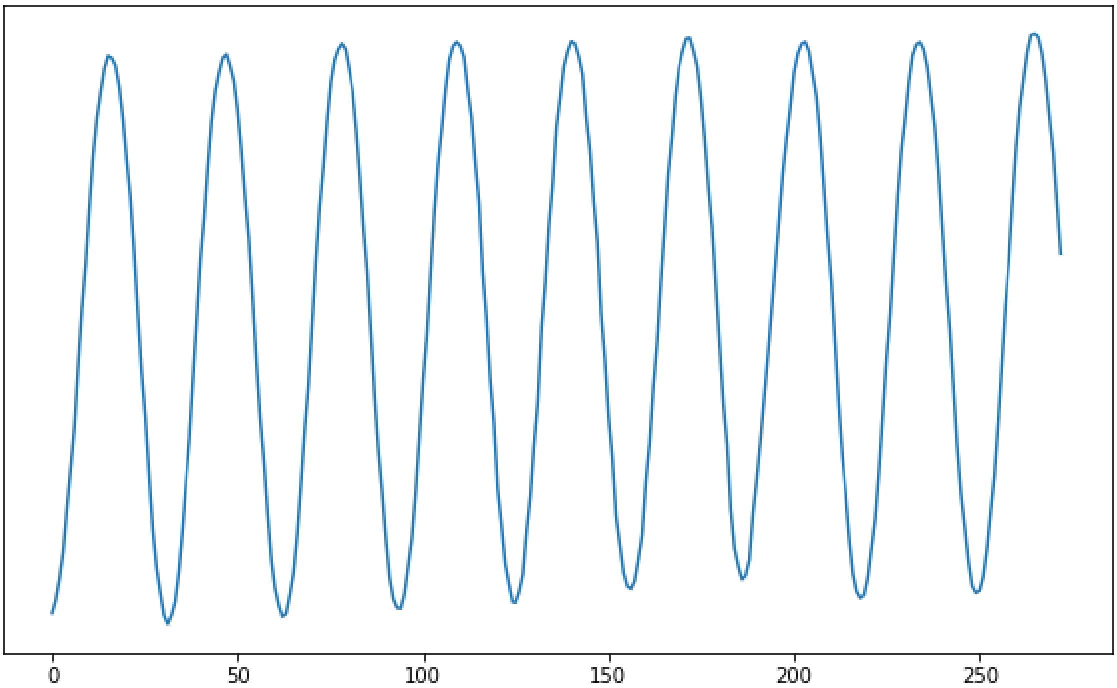
In [9]:

```
# relative magnitutes of the principal components

Z = X*V
xp = np.arange(0,m)/24

plt.figure(figsize=(10, 6))
plt.plot(xp, Z)
plt.yticks([])
plt.show()
```

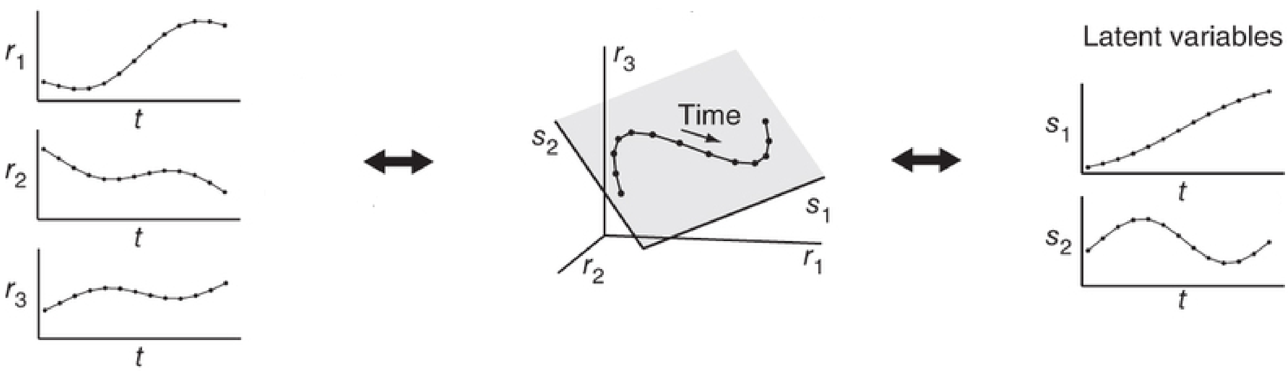


In [10]:

```
## projected onto the first principal component
# 6 dim -> 1 dim (dim reduction)
# relative magnitute of the first principal component

Z = X*V[:,0]

plt.figure(figsize=(10, 6))
plt.plot(Z)
plt.yticks([])
plt.show()
```



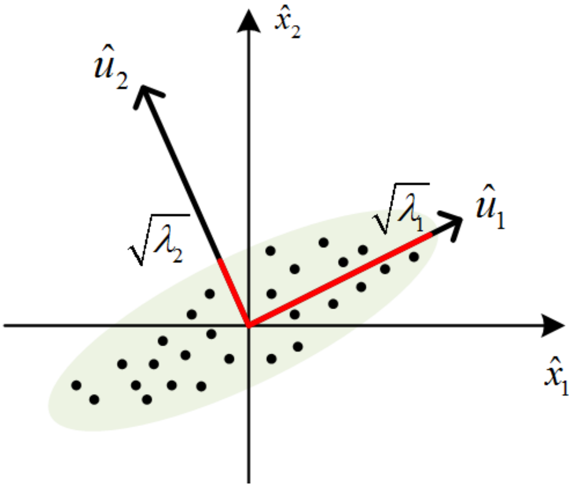
Reference: John P Cunningham & Byron M Yu, Dimensionality reduction for large-scale neural recordings, Nature Neuroscience 17, 1500–1509 (2014)

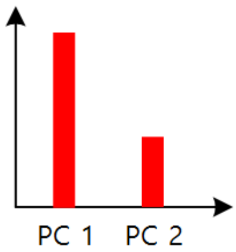


5. PCA Visualization

5.1. Eigenvalues

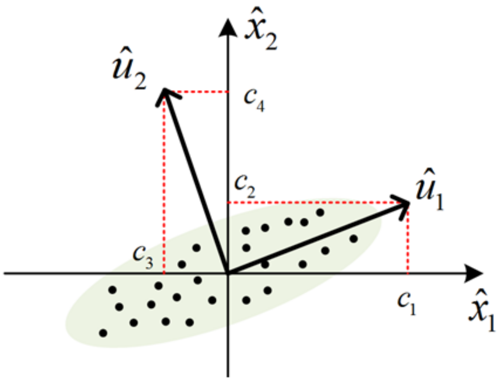
- λ_1, λ_2 indicates variance along the eigenvectors, respectively.
  - The larger eigenvalue is, the more dominante feature (eigenvector) is





5.2. Eigenvectors

- given basis $\{\hat{x}_1, \hat{x}_2\}$ to transformed basis $\{\hat{u}_1, \hat{u}_2\}$



$$[\hat{u}_1 \ \hat{u}_2] = [\hat{x}_1 \ \hat{x}_2] \begin{bmatrix} c_1 & c_3 \\ c_2 & c_4 \end{bmatrix}$$

In [15]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_notebook_to
c.js')
```