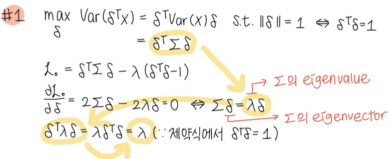
# HW 1

In [31]:
```
!pip install IPython
from IPython.display import Image
```

```
Requirement already satisfied: IPython in c:\users\user\anaconda3\lib\site-packages (7.12.0)
Requirement already satisfied: setuptools>=18.5 in c:\users\user\anaconda3\lib\site-packages
(from IPython) (45.2.0.post20200210)
Requirement already satisfied: backcall in c:\users\user\anaconda3\lib\site-packages (from IP
ython) (0.1.0)
Requirement already satisfied: colorama; sys_platform == "win32" in c:\users\user\anaconda3\l
ib\site-packages (from IPython) (0.4.3)
Requirement already satisfied: pygments in c:\users\user\anaconda3\lib\site-packages (from IP
ython) (2.5.2)
Requirement already satisfied: pickleshare in c:\users\user\anaconda3\lib\site-packages (from
IPython) (0.7.5)
Requirement already satisfied: decorator in c:\users\user\anaconda3\lib\site-packages (from I
Python) (4.4.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in c:\users\user
\anaconda3\lib\site-packages (from IPython) (3.0.3)
Requirement already satisfied: jedi>=0.10 in c:\users\user\anaconda3\lib\site-packages (from
IPython) (0.14.1)
Requirement already satisfied: traitlets>=4.2 in c:\users\user\anaconda3\lib\site-packages (f
rom IPython) (4.3.3)
Requirement already satisfied: wcwidth in c:\users\user\anaconda3\lib\site-packages (from pro
mpt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->IPython) (0.1.8)
Requirement already satisfied: parso>=0.5.0 in c:\users\user\anaconda3\lib\site-packages (fro
m jedi>=0.10->IPython) (0.5.2)
Requirement already satisfied: ipython-genutils in c:\users\user\anaconda3\lib\site-packages
(from traitlets>=4.2->IPython) (0.2.0)
Requirement already satisfied: six in c:\users\user\anaconda3\lib\site-packages (from traitle
ts>=4.2->IPython) (1.14.0)
```

In [32]:
```
Image("C:/Users/User/Desktop/KakaoTalk_20210819_104918768.jpg")
```

Out[32]:



#1 $\max_{\delta} Var(\delta^T X) = \delta^T Var(X)\delta$ s.t. $\|\delta\| = 1 \Leftrightarrow \delta^T\delta = 1$

$= \delta^T \Sigma \delta$

$\mathcal{L}_0 = \delta^T\Sigma\delta - \lambda(\delta^T\delta - 1)$

$\frac{\partial \mathcal{L}_0}{\partial \delta} = 2\Sigma\delta - 2\lambda\delta = 0 \Leftrightarrow \Sigma\delta = \lambda\delta$

Σ의 eigenvalue

Σ의 eigenvector

$\delta^T\lambda\delta = \lambda\delta^T\delta = \lambda$ (∵ 제약식에서 $\delta^T\delta = 1$)

# HW 2

In [1]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
import seaborn as sns
```

### Bounus : PCA examples

### Eigenfaces : PCA as feature selector

In [3]:
```
from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/5976015
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

In [4]:
```
# number of features (pixels)
62*47
```

Out[4]:
```
2914
```

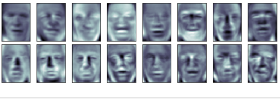
Let's take a look at the principal axes that span this dataset. Because this is a large dataset, we will use RandomizedPCA—it contains a randomized method to approximate the first *N* principal components much more quickly than the standard PCA estimator, and thus is very useful for high-dimensional data (here, a dimensionality of nearly 3,000). We will take a look at the first 150 components:

In [5]:
```
pca = PCA(n_components=150, svd_solver='randomized')
pca.fit(faces.data)
```

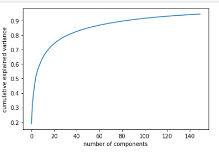
Out[5]:
```
PCA(copy=True, iterated_power='auto', n_components=150, random_state=None,
    svd_solver='randomized', tol=0.0, whiten=False)
```

Now, let's take a look at first 24 basis components/features of images. 처음 몇 개의 eigenface들은 빛의 각도 등 전반적인 feature라면 뒤로 갈수록 더 자세한 feature들(눈코입)의 map이 뽑아짐을 확인할 수 있다.

In [6]:
```
fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(pca.components_[i].reshape(62, 47), cmap='bone')
```

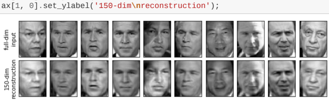


In [7]:
```
# how many components to choose?
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



In [8]:
```
# Compute the components and projected faces
pca = PCA(150, svd_solver = 'randomized').fit(faces.data)
components = pca.transform(faces.data)
projected = pca.inverse_transform(components) # reconstruction
```

In [9]:
```
# Plot the results
fig, ax = plt.subplots(2, 10, figsize=(10, 2.5),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    ax[0, i].imshow(faces.data[i].reshape(62, 47), cmap='binary_r')
    ax[1, i].imshow(projected[i].reshape(62, 47), cmap='binary_r')

ax[0, 0].set_ylabel('full-dim\ninput')
ax[1, 0].set_ylabel('150-dim\nreconstruction');
```



### HW : PCA from scratch

- iris 데이터에 numpy만을 사용해서 PCA를 해봅시다!
- We can find PCA makes visualization easier!

In [10]:
```
df = pd.read_csv('https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.cs
v')
df.head()
```

Out[10]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

In [11]:
```
X = df.iloc[:,:-1]
label = df.iloc[:,-1]
X.head()
```

Out[11]:

| | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 |

#### Using Covariance Matrix

In [12]:
```
# Step 1. Center Data
X_scaled = StandardScaler().fit_transform(X)
X_scaled[:5]
```

Out[12]:
```
array([[-0.90068117,  1.03205722, -1.3412724 , -1.31297673],
       [-1.14301691, -0.1249576 , -1.3412724 , -1.31297673],
       [-1.38535265,  0.33784833, -1.39813811, -1.31297673],
       [-1.50652052,  0.10644536, -1.2844067 , -1.31297673],
       [-1.02184904,  1.26346019, -1.3412724 , -1.31297673]])
```

In [14]:
```
# Step 2. Compute Covariance Matrix
cov_matrix = np.cov(X_scaled.T)
cov_matrix
```

Out[14]:
```
array([[ 1.00671141, -0.11010327,  0.87760486,  0.82344326],
       [-0.11010327,  1.00671141, -0.42333835, -0.358937  ],
       [ 0.87760486, -0.42333835,  1.00671141,  0.96921855],
       [ 0.82344326, -0.358937  ,  0.96921855,  1.00671141]])
```

In [20]:
```
# Step 3. Eigenvalue decomposition
eigvals, eigvecs = np.linalg.eig(cov_matrix)
print("Eigenvalues:", eigvals)
print("")
print("Eigenvectors:", eigvecs)
```

```
Eigenvalues: [2.93035378 0.92740362 0.14834223 0.02074601]

Eigenvectors: [[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]
```

In [21]:
```
# Ratio of explained variance per PC
explained_variances = []
for i in range(len(eigvals)):
    explained_variances.append(eigvals[i] / np.sum(eigvals))

print(np.sum(explained_variances), '\n', explained_variances)
```

```
0.9999999999999999
 [0.7277045209380132, 0.23030523267680636, 0.03683831957627389, 0.0051519268089063935]
```

첫 번째, 두 번째 PC가 이미 variance의 95% 이상을 설명함!

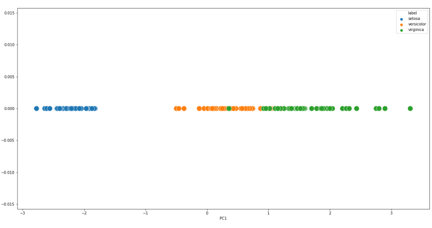
In [24]:
```
# Visualization (Embedding)
pc1 = np.dot(X_scaled, eigvecs[:,0])
pc2 = np.dot(X_scaled, eigvecs[:, 1])
res = pd.DataFrame(pc1, columns=['PC1'])
res['PC2'] = pc2
res['label'] = label
res.head()
```

Out[24]:

| | PC1 | PC2 | label |
|---|---|---|---|
| **0** | -2.264542 | -0.505704 | setosa |
| **1** | -2.086426 | 0.655405 | setosa |
| **2** | -2.367950 | 0.318477 | setosa |
| **3** | -2.304197 | 0.575368 | setosa |
| **4** | -2.388777 | -0.674767 | setosa |

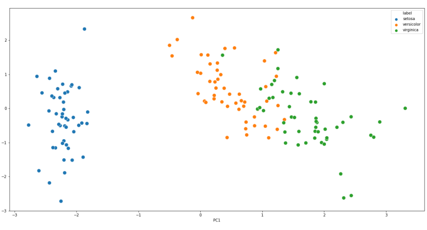
In [25]:
```
# Projection on 1-dim subspace
plt.figure(figsize=(20, 10))
sns.scatterplot(res['PC1'], [0] * len(res), hue=res['label'], s=200)
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x193cdc65a08>
```



In [26]:
```
# Projection on 2-dim subspace
plt.figure(figsize=(20, 10))
sns.scatterplot(res['PC1'], res['PC2'], hue=res['label'], s=100)
```

Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x193cf8a0cc8>
```



#### Shortcut

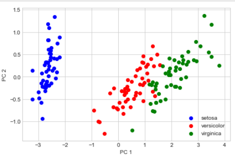
In [27]:
```
from sklearn.decomposition import PCA as sklearnPCA

sklearn_pca = sklearnPCA(n_components=2)
projection = sklearn_pca.fit_transform(X, y=label)

sklearn_pca.explained_variance_ratio_
```

Out[27]:
```
array([0.92461621, 0.05301557])
```

In [28]:
```
sklearn_pca.components_
```

Out[28]:
```
array([[ 0.36158968, -0.08226889,  0.85657211,  0.35884393],
       [ 0.65653988,  0.72971237, -0.1757674 , -0.07470647]])
```

In [29]:
```
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('setosa', 'versicolor', 'virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(projection[label==lab, 0],
                    projection[label==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('PC 1')
    plt.ylabel('PC 2')
    plt.legend(loc='lower right')
    plt.tight_layout()
    plt.show()
```



In [ ]:
```
```