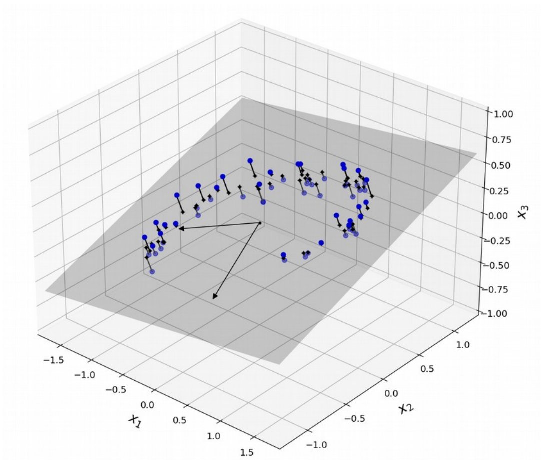
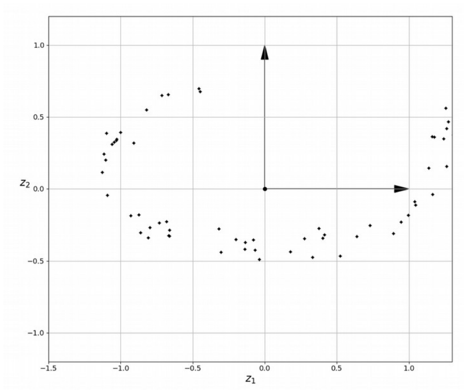
Q1-1. SVD & PCA

```python
import sys
assert sys.version_info >= (3, 5)

import sklearn
assert sklearn.__version__ >= "0.20"

import numpy as np
import os

import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d

from mpl_toolkits.mplot3d import Axes3D

##################################################################################################################################################
### Preparation ##################################################################################################################################
##################################################################################################################################################
np.random.seed(42)

mpl.rc('axes', labelsize=14)        # 좌표축 이름 크기 정의
mpl.rc('xtick', labelsize=12)       # X 축 단위 이름 크기 정의
mpl.rc('ytick', labelsize=12)       # Y 축 단위 이름 크기 정의

# 그래프 결과를 저장한 경로 정의
PROJECT_ROOT_DIR = '.'
CHAPTER_ID = 'dim_reduction'
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, 'images', CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

# 그래프 결과를 저장하는 함수
def save_fig(fig_id, tight_layout=True, fig_extension='png', resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + '.' + fig_extension)
    print('Save Image ', fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

import warnings
warnings.filterwarnings(action='ignore', message='^internal gelsd')

##################################################################################################################################################
### Random Dataset Generation ####################################################################################################################
##################################################################################################################################################
np.random.seed(4)
m = 60
w1, w2 = 0.1, 0.3
noise = 0.1

# 3 개의 Feature 로 구성된 랜덤 데이터셋 생성
angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
X = np.empty((m, 3))
X[:, 0] = np.cos(angles) + np.sin(angles) / 2 + noise * np.random.randn(m) / 2
X[:, 1] = np.sin(angles) * 0.7 + noise * np.random.randn(m) / 2
X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + noise * np.random.randn(m)

##################################################################################################################################################
### Dimensionality Reduction #####################################################################################################################
##################################################################################################################################################
### SVD  기반 차원 축소 ###
X_centered = X - X.mean(axis=0)     #  데이터셋 표준화

U, s, Vt = np.linalg.svd(X_centered)     # SVD = Principle Component (Eigen Vector) 를 구하기 위한 연산 과정
# U = Left Singlular Vector = A * A^T  연산 결과
# Vt = Right Singular Vector = A^T * A  연산 결과  = SVD 를 통해 산출한 Principle Components
# s = A * A^T 와 A^T * A  대한 Eigen Value

c1 = Vt.T[:, 0]      # SVD 를 통해 산출한 1st Principle Component
c2 = Vt.T[:, 1]      # SVD 를 통해 산출한 2nd Principle Component

m, n = X.shape

S = np.zeros(X_centered.shape)
S[:n, :n] = np.diag(s)

print(np.allclose(X_centered, U.dot(S).dot(Vt)))     # SVD  연산 결과와 원본 데이터셋이 서로 유사한지 확인함
                                                    # U * S * Vt =  원본 데이터셋으로 재구성 결과

W2 = Vt.T[:, :2]            # 1st Principle Component 와 2nd Principle Component 를 선택함
X2D_using_svd = X_centered.dot(W2)     # 1st  PC 와 2nd  PC 에 표준화된 데이터셋을 투영함
                                      # --> 1st  PC 와 2nd  PC 를 새로운 Feature 로 설정하여 데이터셋을 재구성함

### PCA  기반 차원 축소 ###
pca = PCA(n_components=2)       #  데이터셋에 대해  2 개의 최상 Principle Component 를 산출하는 PCA  객체
                                # --> 1st  PC 와 2nd  PC 를 새로운 Feature 로 설정하여 데이터셋을  2 차원에 재구성함

X2D = pca.fit_transform(X)      #  재구성된 데이터셋을 생성함 (sklearn API 는 표준화 과정을 내장하고 있음 )

print(X2D[:5])
print(X2D_using_svd[:5])

print(np.allclose(X2D, -X2D_using_svd)) # PCA  기반 데이터셋 재구성 결과와  SVD  기반 데이터셋 재구성 결과가 유사한지 확인함

X3D_inv = pca.inverse_transform(X2D)     #  데이터셋 원상 복구 : 기존 Feature 로 재구성된 데이터셋을 재투여함

print(np.allclose(X3D_inv, X))          #  원본 데이터셋와 복구된 데이터셋이 서로 1e-8  이내로 유사한지 확인함
                                        # --> False ( 유사하지 않음 ) /  이유 : 3D-2D Reprojection  과정에서 데이터셋에 대한 정보 손실이 발생함

print(np.mean(np.sum(np.square(X3D_inv - X), axis=1)))  #  원본 데이터셋과 복구도니 데이터셋 사이의 Error  비율 출력

X3D_inv_using_svd = X2D_using_svd.dot(Vt[:2, :])     # SVD  기반 재구성 데이터셋을 원본 영역으로 재투영하여 복구시킴

print(np.allclose(X3D_inv_using_svd, X3D_inv - pca.mean_))
# SVD  기반 데이터셋의 원본 결과와  PCA  기반 데이터셋의 원본 결과가 서로 유사한지 확인함
# 평균값을 뺀 결과를 사용한 이유는 sklearn PCA API 가 자동으로 표준화를 수행하는 것을 반영함

print(pca.components_)      # PCA 로 구한  2 개의 최상 Principle Component 를 출력함

print(Vt[:2])               # SVD 로 구한  2 개의 최상 Principle Component 를 출력함

print(pca.explained_variance_ratio_)      # PCA 의 Principle Component 에 대한 각각의 Variance  출력함

print(1 - pca.explained_variance_ratio_.sum())  # PCA 에 의한 Variance  손실을 출력함

print(np.square(s) / np.square(s).sum())  # SVD 의 Principle Component 에 대한 각각의 Variance  출력함

##################################################################################################################################################
### Dimensionality Reduction Plotting ############################################################################################################
##################################################################################################################################################

# Reprojection 을 표현하기 위한 3D Arrow  클래스
class Arrow3D(FancyArrowPatch):
    # 3D Arrow  객체 초기화
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0, 0), (0, 0), *args, **kwargs)
```

```python
        self._verts3d = xs, ys, zs

    # 3D Arrow 를 그리는 함수
    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0], ys[0]), (xs[1], ys[1]))
        FancyArrowPatch.draw(self, renderer)

axes = [-1.8, 1.8, -1.3, 1.3, -1.0, 1.0]     # 3 차원을 표현하기 위한 범위

x1s = np.linspace(axes[0], axes[1], 10)      # 1 번 Axis 에 대한 범위 설정
x2s = np.linspace(axes[2], axes[3], 10)      # 2 번 Axis 에 대한 범위 설정
x1, x2 = np.meshgrid(x1s, x2s)

C = pca.components_      # 2 개의 최상 Principle Component  저장함
R = C.T.dot(C)           # Principle Component 간 내적을 통해 재투영을 위한 영역을 설정함
z = (R[0, 2] * x1 + R[1, 2] * x2) / (1 - R[2, 2])    # 2 개의 최상 Principle Component 로 구성된 2D  영역

fig = plt.figure(figsize=(6, 3.8))
ax = fig.add_subplot(111, projection='3d')

X3D_above = X[X[:, 2] > X3D_inv[:, 2]]       #  새로운 2D  영역 위에 배치된 데이터
X3D_below = X[X[:, 2] <= X3D_inv[:, 2]]      #  새로운 2D  영역 아래에 배치된 데이터

ax.plot(X3D_below[:, 0], X3D_below[:, 1], X3D_below[:, 2], "bo", alpha=0.5)

ax.plot_surface(x1, x2, z, alpha=0.2, color="k")     # PCA 로 산출한 2 개의 최상 Principle Component 로 구성된 2D  영역을 그림
np.linalg.norm(C, axis=0)
# 1st Principle Component 를 표현하는 화살표를 그림
ax.add_artist(Arrow3D([0, C[0, 0]],[0, C[0, 1]],[0, C[0, 2]], mutation_scale=15, lw=1, arrowstyle="-|>", color="k"))
# 2nd Principle Component 를 표현하는 화살표를 그림
ax.add_artist(Arrow3D([0, C[1, 0]],[0, C[1, 1]],[0, C[1, 2]], mutation_scale=15, lw=1, arrowstyle="-|>", color="k"))
ax.plot([0], [0], [0], "k.")

# 원본 데이터셋과 재투영된 데이터셋 사이에 선을 그림
for i in range(m):
    if X[i, 2] > X3D_inv[i, 2]:
        ax.plot([X[i][0], X3D_inv[i][0]], [X[i][1], X3D_inv[i][1]], [X[i][2], X3D_inv[i][2]], "k-")
    else:
        ax.plot([X[i][0], X3D_inv[i][0]], [X[i][1], X3D_inv[i][1]], [X[i][2], X3D_inv[i][2]], "k-", color="#505050")

# 원본 데이터셋을 3D  영역에 그리고 / 재투영된 데이터셋을 Principle Component 로 구성된 2D  영역에 그림
ax.plot(X3D_inv[:, 0], X3D_inv[:, 1], X3D_inv[:, 2], "k+")
ax.plot(X3D_inv[:, 0], X3D_inv[:, 1], X3D_inv[:, 2], "k.")
ax.plot(X3D_above[:, 0], X3D_above[:, 1], X3D_above[:, 2], "bo")
ax.set_xlabel("$x_1$", fontsize=18, labelpad=10)
ax.set_ylabel("$x_2$", fontsize=18, labelpad=10)
ax.set_zlabel("$x_3$", fontsize=18, labelpad=10)
ax.set_xlim(axes[0:2])
ax.set_ylim(axes[2:4])
ax.set_zlim(axes[4:6])

save_fig("dataset_3d_plot")      #  그래프 출력 결과 저장
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')

# Principle Component 로 재구성된 / 재투영된 데이터셋을 2D  영역에 그림
ax.plot(X2D[:, 0], X2D[:, 1], "k+")
ax.plot(X2D[:, 0], X2D[:, 1], "k.")
ax.plot([0], [0], "ko")
ax.arrow(0, 0, 0, 1, head_width=0.05, length_includes_head=True, head_length=0.1, fc='k', ec='k')    # Principle Component 에 대한 화살표를 그림
ax.arrow(0, 0, 1, 0, head_width=0.05, length_includes_head=True, head_length=0.1, fc='k', ec='k')    # Principle Component 에 대한 화살표를 그림
ax.set_xlabel("$z_1$", fontsize=18)
ax.set_ylabel("$z_2$", fontsize=18, rotation=0)
ax.axis([-1.5, 1.3, -1.2, 1.2])
ax.grid(True)
save_fig("dataset_2d_plot")

plt.show()
```



PCA 를 통한 차원 축소로 3D 데이터를
2 개의 상위 Principle Component 로 구성된 2D Plane 으로 투영한 결과



PCA 를 통한 차원 축소로 3D 데이터를
2 개의 상위 Principle Component 로 구성된 2D Plane 된 2D 데이터 분포