3. $||x_i - x_j||_2 = ||z_i - z_j||_2$. Lets take a look at $||z_i - z_j||_2$:

$$\begin{aligned}||z_i - z_j|| &= \|U^T(x_i - \mu_x) - U^T(x_j - \mu_x)\| \\ &= \|U^T x_i - U^T \mu_x - U^T x_j + U^T \mu_x\| \\ &= \|U^T\| \cdot \|x_i - x_j\| = \|x_i - x_j\|\end{aligned}$$

?

-2

■

1.2.2

Let $U_d \in \mathbb{R}^{D\times d}$ be a fully ranked matrix (with $d \leq D$). Since U is fully ranked, we can use SVD and denote $U_d = U'\Sigma' V'^T$ where $U' \in \mathbb{R}^{D\times D}$ $\Sigma' \in \mathbb{R}^{D\times d}$ $V' \in \mathbb{R}^{d\times d}$. We will notice that $V'^T V' = I_d$, and that Σ'^{-1} exists (every diagonal matrix has an invert)
We will denote $M = V'\Sigma'^{-1}, O = U_d M$ and we will take a look at $O^T O$:

$$\begin{aligned}O^T O &= M^T U_d^T U_d M \\ &= (\Sigma^{-1})^T V'^T V' \Sigma^T U'^T U' \Sigma' V'^T V' \Sigma'^{-1} = \\ &=^* (\Sigma^{-1})^T I_d \Sigma^T I_d \Sigma' I_d \Sigma^{-1} = I_d\end{aligned}$$

-1

(*) $V'^T V' = I_d, U'^T U' = I_D$ ■

Σ_x is not square

You should have used the compact SVD instead.

1.2.6

$X \in \mathbb{R}^{D\times N}$ where $D > N$.
A tight upper bound to the number of non zero eigenvalues would be N.

Lets take an example $I_N \in \mathbb{R}^{D\times N} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 1 \\ 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$

In I_D, there are N eigenvectors (each one of the columns is an ~~eigenvector~~)

Now, Since the $Rank(X) \leq min(N, D) = N$, it means that X can be spread with ~~D~~ N independent vectors, so there can not be more then N eigenvalues greater then 0. ■

What about the second part?

-2

1.2.7

Let $A \in \mathbb{R}^{D\times N}$ and we have the following minimization problem:

$$\begin{cases} \min_{M\in\mathbb{R}^{D\times N}} \|A-M\|_F^2 \\ s.t\ rank(M) \leq d \end{cases}$$

A is not square

We will notice, that because $rank(M) = d$ then $d \leq \min(D,N)$ and therefore we have two matrices $B \in \mathbb{R}^{D\times d}, C \in \mathbb{R}^{d\times N}$ with ranks d such that $M = BC$. We also saw in class the the solution of this minimization problem is $C = B_d^T A$ where B_d are eigenvectors of A, and we have only d of them and all the rest of B's columns are 0.
We also know that every matrix $M \in \mathbb{R}^{D\times N}$ can be expressed as $M = U\Sigma V^T$. Because M's rank is d, then we will have the truncated version $M = U_d\Sigma_d V_d^T$ where again U_d are matrix with d eigenvectors of A, Σ_d has d eigenvalues along the diagonal and the rest of the diagonal is 0 (if not truncated) and V_d^T is a row-vector matrix where we have again d vecotrs which are non-zero.

?

+2

To be honest, I tried and tried, and didn't figure out how to combine both these points into a final solution. I tried to transform B_dC_d into a representation of $U_d\Sigma_d V_d^T$ but unfortunately couldn't find the right way.

Prove: $k(x_i,x_j) = (1+x_i^T x_j)^2$ is a kernel function.

Lets write :

? ?

$$\phi(x) = 1 + 2x_i^T x_j + x_i^2 = \begin{bmatrix} 1 \\ \sqrt{2}x \\ x^2 \end{bmatrix}$$

Now, lets show that at $k(x_i,x_j) =< \phi(x_i), \phi(x_j) >$

$$\begin{aligned} k(x_i,x_j) &= (1+x_i^T x_j)^2 \\ &= (1+x_i^T x_j)(1+x_i^T x_j) \\ &= 1 + 2x_i^T x_j + (x_i^T x_j)^2 \end{aligned}$$

-2

-2

$$= (1 + x_i^T x_j)(1 + x_i^T x_j)$$
$$= 1 + 2x_i^T x_j + (x_i^T x_j)^2$$
$$2? \; = < \begin{bmatrix} 1 \\ \sqrt{2}x_i \\ x_i^2 \end{bmatrix}, \begin{bmatrix} 1 \\ \sqrt{2}x_j \\ x_j^2 \end{bmatrix}$$
$$=< \phi(x_i), \phi(x_j) >$$

2.1.7 OOS extension

Let K_x be the kernel matrix obtained from the training set $\chi = \{x_i \in \mathbb{R}^D\}_{i=1}^N$, let $Z \in \mathbb{R}^{d\times N}$ be the low-dimentional representation obtained by applying KPCA, that is: $Z = \Sigma_d V_d^T$ and let $X^* \in \mathbb{R}^{D\times N}$ be a set of new unseen data-points.
We will denote $\Phi^* = \phi(X^*)$ and $K_x^* = \Phi^T \Phi$.
Then from what we learned in the lecture, we can represent the KPCA of the OOS extension applied on X^* as:

$$Z^* = \Sigma_d^{-1} V_d^T \widetilde{\Phi^T} \widetilde{\Phi^*} \cancel{=} \Sigma_d^{-1} V_d^T \widetilde{K_x^*}$$

-2

Since $\widetilde{K_x^*} = J(\underbrace{k_x - \frac{1}{N} K_x 1_N})$

$\mathbb{R}^{N\times N^*} \neq \mathbb{R}^{N^*}$

$$Z^* = \Sigma_d^{-1} V_d^T \widetilde{K_x^*} \cancel{=} \Sigma_d^{-1} V_d^T J(k_x - \frac{1}{N} K_x 1_N)$$

■

```python
class PCA:
    def __init__(self, d):
        self.d     = d
        self.vMean = None
        self.mUd   = None
        self.vSig  = None

    def Fit(self, mX):
        '''
        Learns model's parameters
        Args:
            mX - Input training data, mX.shape = (D, N)
        Output:
            self
        '''

        svds_values = svds(mX, self.d)
        eigen_vectors, eigen_values = svds_values[0], svds_values[1]

        idx = eigen_values.argsort()[::-1]
        self.mUd   = eigen_vectors[:,idx]
        self.vSig  = eigen_values[idx]
        self.vMean = svds_values[2]

```

-2

?

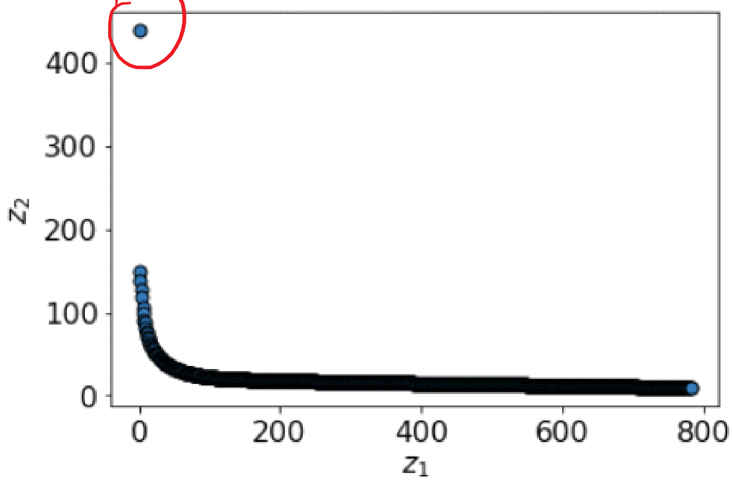
1. **You need to remove the mean first**
2. **What?!**

2. What?!

```python
 _ , eigenvalues, _ = svds(mX, mX.shape[0] - 1)

eigenvalues = np.flip(eigenvalues) #sort from higher to lower

plt.scatter(range(len(eigenvalues)), eigenvalues, s=50, edgecolor='k')
plt.xlabel('$z_1$')
plt.ylabel('$z_2$')
plt.show()
```



need to remove the mean

```python
class KPCA:
    def __init__(self, d, k):
        self.d = d
        self.k = k
        pass

    def Fit(self, mX):
        '''
        Learns model's parameters
        Args:
            mX - Input training data, mX.shape = (D, N)
        Output:
            self
        '''

        N = mX.shape[1]
        kernel = self.k(mX, mX)
        J = np.identity(N) - np.ones((N , N)) / N
        self.mX = mX
        self.K = kernel @ J

        svds_values = svds(self.K, self.d)
        eigen_vectors, eigen_values = svds_values[0], svds_values[1]

        idx = eigen_values.argsort()[::-1]
        self.mVd   = eigen_vectors[:,idx]
        self.vSig  = np.sqrt(eigen_values[idx])

    def Encode(self, mXstar):
        '''
        Apply (out of sample) encoding
        Args:
            mXstar - Input data,                              mX.shape = (D, Nstar)
        Output:
            mZ     - Low-dimensional representation (embeddings), mZ.shape = (d, Nstar)
        '''
        return np.linalg.inv(np.diag(self.vSig)) @ self.mVd.T @ self.k(self.mX, mXstar)
```

$\tilde{K} = J K J$

$\tilde{K} = V \Sigma^2 V^T$

3

```
        Output:
            mZ      - Low-dimensional representation (embeddings), mZ.shape = (d, Nstar)
        '''
        return np.linalg.inv(np.diag(self.vSig)) @ self.mVd.T @ self.k(self.mX, mXstar)

```

```
from scipy.spatial.distance import cdist
def mat_mul_kernel(mX1, mX2):
  return mX1.T @ mX2

def polynomial_kernel(mX1, mX2, p=2):
  return (1 + mX1.T @ mX2) ** p

def gaus_kernel(mX1, mX2):
  distances_matrix = cdist(mX1.T, mX2.T)
  sigma = np.mean(distances_matrix)
  return np.exp(-distances_matrix / (2 * sigma * sigma))
```

−2

1. **sigma must be a fixed value**
2. **dist_mat ** 2**