This assignment was incredibly confusing given that the assignment instructions regarding matrix sizes and shapes did not match the notes. It made it it especially confusing because the implementation was verbatim to the notes pseudo code. That is, I was able to explicitly write out C = np.dot(X, X.T) where X was the “Data”. Also O was able to do Y = np.dot(U.T, X) as well without any shape incopataiblities. This was so confusing to me because it worked this way using the assignments shapes and sizes. That is teh “evectors” array AKA the U_k matrix is M x numpc and the “representation” AKA the U matrix is numpc x N in the assignment(where numpc is k). So by using these definitions , which dont really make sense in regards to my conceptual understanding of the algorithm, it was straightforward to implement the pseudo code.

However when I tried using the shapes and sizes from the notes tat make more sense, i.e., evectors AKA U = NxK and representation AKA Y = U*X I had to really change how I did the matrix operations and reshape alot of the arrays. So this whole assignment was incredibly confusing.

For example: lets say we want k= numpc = 4 PCs.

Assignment Logic:

- C = 1000 x 1000
- Evectors / U = M x numpc = 1000 x 4
- Representation / Y = numpc x N = 4 x 784

Notes Logic:

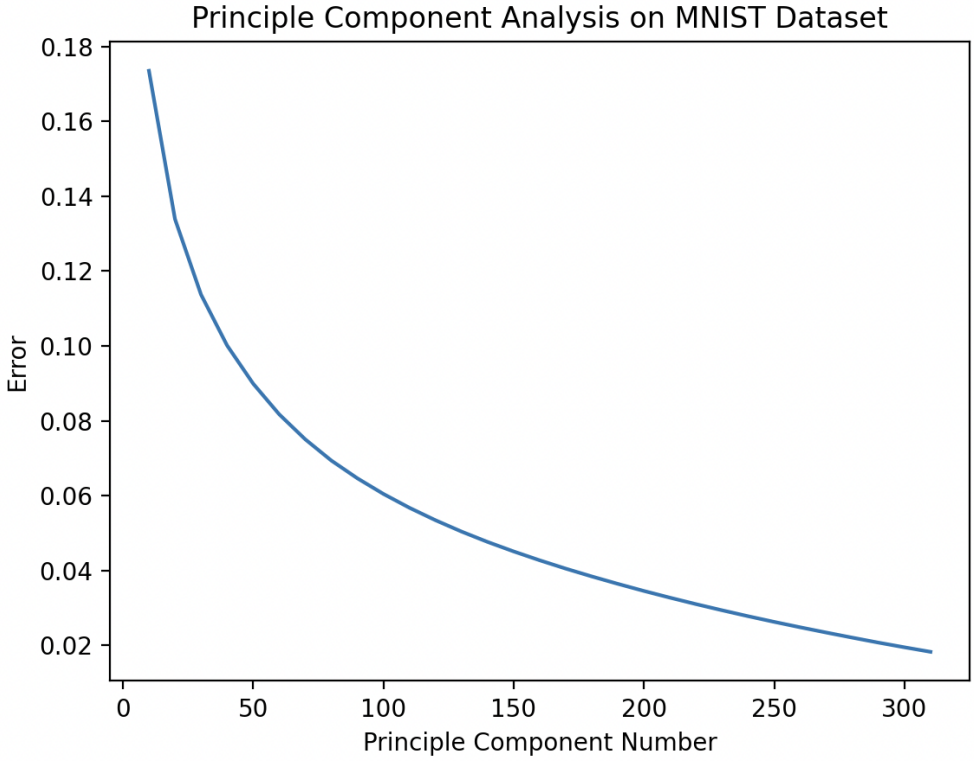
- C = 784 x 784
- Evectors / U= N x numpc = 784 x 4
- Representation / Y = numpc x M = 4 x 1000

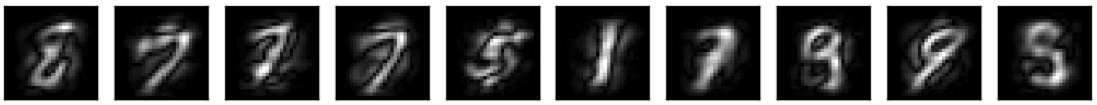
The covariance matrix makes no sense being 1000 x 1000 but that's the only way to get eigenvalues and vectors in that shape and thus evectors and representation vectors in the shape outlined by the assignment

The only way I was able to get the implementation to match the pseudo code was by changing the shape of the data or X matrix from 1000 x 784 to 784 x 1000 in the beginning.

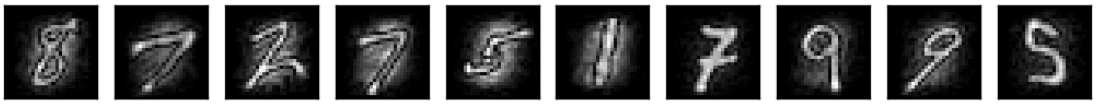
Frustrating that the assignment instructions add confusion to an already complex and abstract topic…

Interestingly though, there seems to be zero difference between which logic is used….

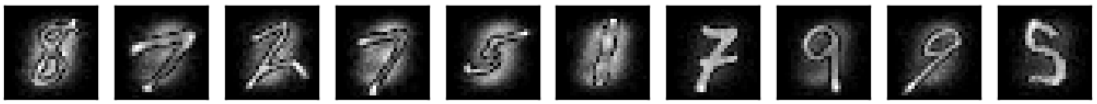




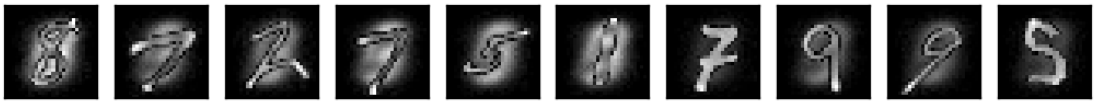
PC 20 Error 0.15



PC 120 Error 0.06



PC 220 Error 0.03



PC 320 Error 0.02

```
def princomp(Data,numpc=0):
    M = Data.shape[0] #number of samples
    N = Data.shape[1] #number of features

    #Step1: mean normalization
    for i in range(M):
        xi = Data[i,:] - ((1/M) * np.sum(Data, axis=0))
        Data[i] = xi

    test = np.sum(Data[:,]) #should be zero but its not
    print("test for mean centered = ", test)
    Data = Data.T
    #Step2: eigendecomposition
    C = (1/M) * dot(Data, Data.T)
    lmbda, v = linalg.eig(C)

    #Step3: sort the eigenvectors and eigenvalues in the descending order of eigenvalues
    lmbda = np.real(lmbda)
    maxindices = np.argpartition(lmbda, -numpc)[-numpc:]

    #Step4: sort eigenvectors according to the sorted eigenvalues
    evectors = np.zeros((N,numpc))
    for i in range(len(maxindices)):
        evectors[:,i] = v[:,maxindices[i]]

    print("evectors shape NxK(U_k[784 x k])= ", evectors.shape)

    #Step5: generate the 'numpc' dimensional representation
    representation = dot(evectors.T, Data)
    print("representation shape KxM (Y[kx1000])= ",representation.shape)

    return evectors,representation
```