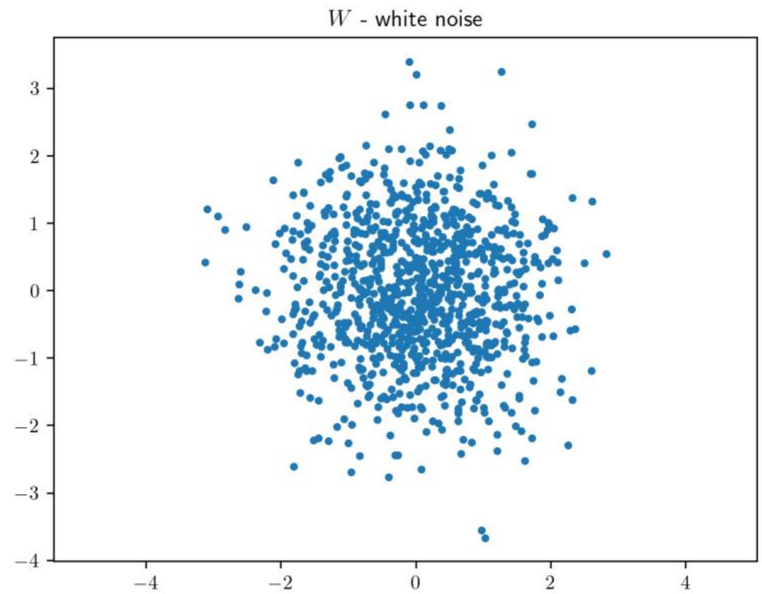
Report Lab 3 – Misha Tsysin, 0033922418

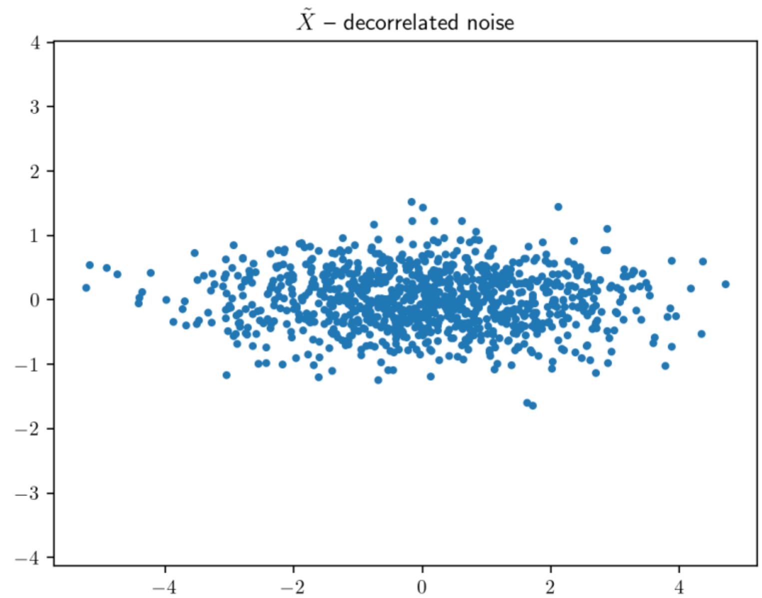
Exercise: Generating Gaussian random vectors

The required scatterplots are:

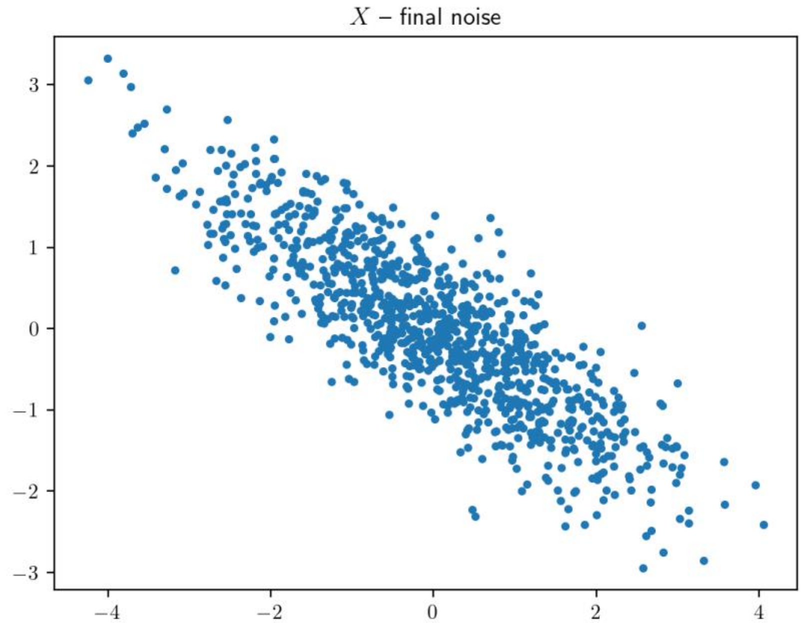
White original noise:



Scaled noise decorrelated:



Rotated noise with given covariance:



Covariance Estimation and Whitening

The theoretical value is:

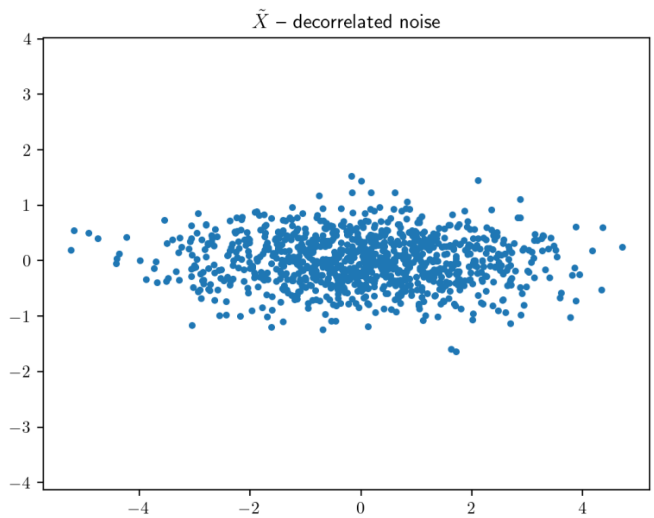
$$R_x = \begin{bmatrix} 2 & -1.2 \\ -1.2 & 1 \end{bmatrix}$$

The estimated matrix is:

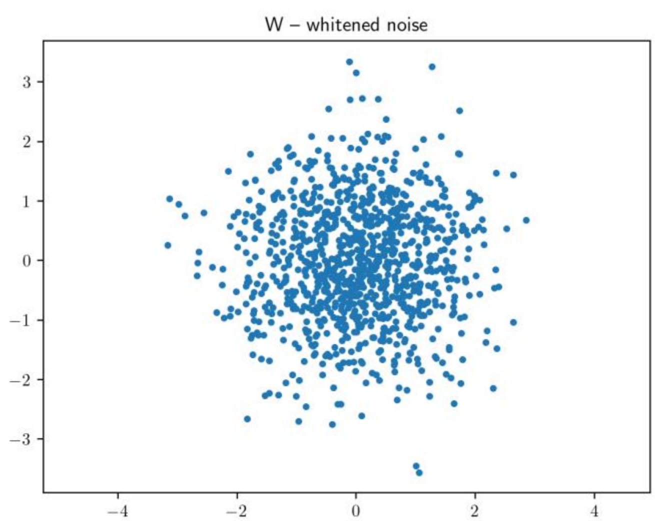
$$\widehat{R_x} = \begin{bmatrix} 2.04 & -1.23 \\ -1.23 & 1.03 \end{bmatrix}$$

The two required scatter plots:

For decorrelated noise:



For the final whitened noise:

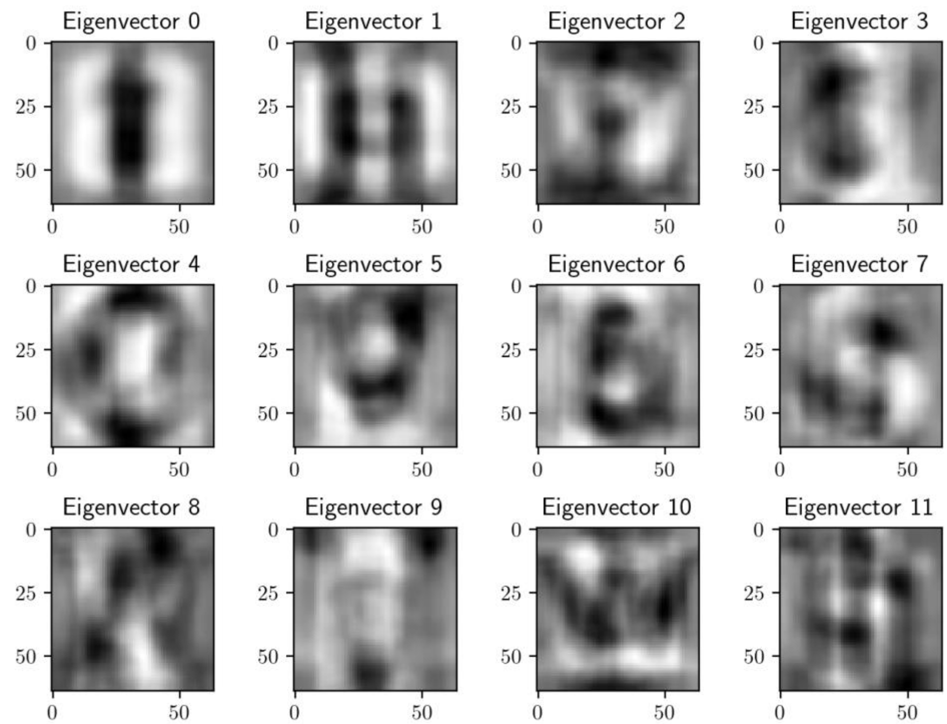


The final covariance of whitened noise is:

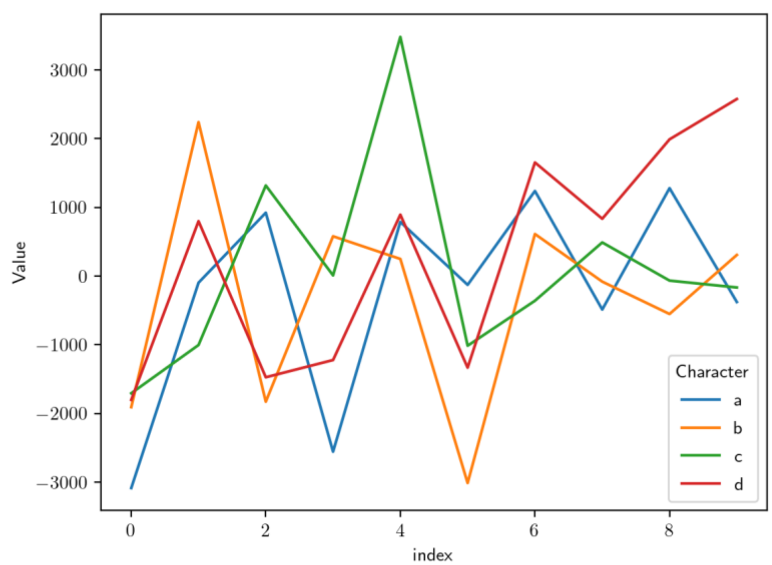
$$\widehat{R_W} = \begin{bmatrix} 1.027 & -0.012 \\ -0.012 & 1.038 \end{bmatrix}$$

Eigenimages, PCA, and Data Reduction

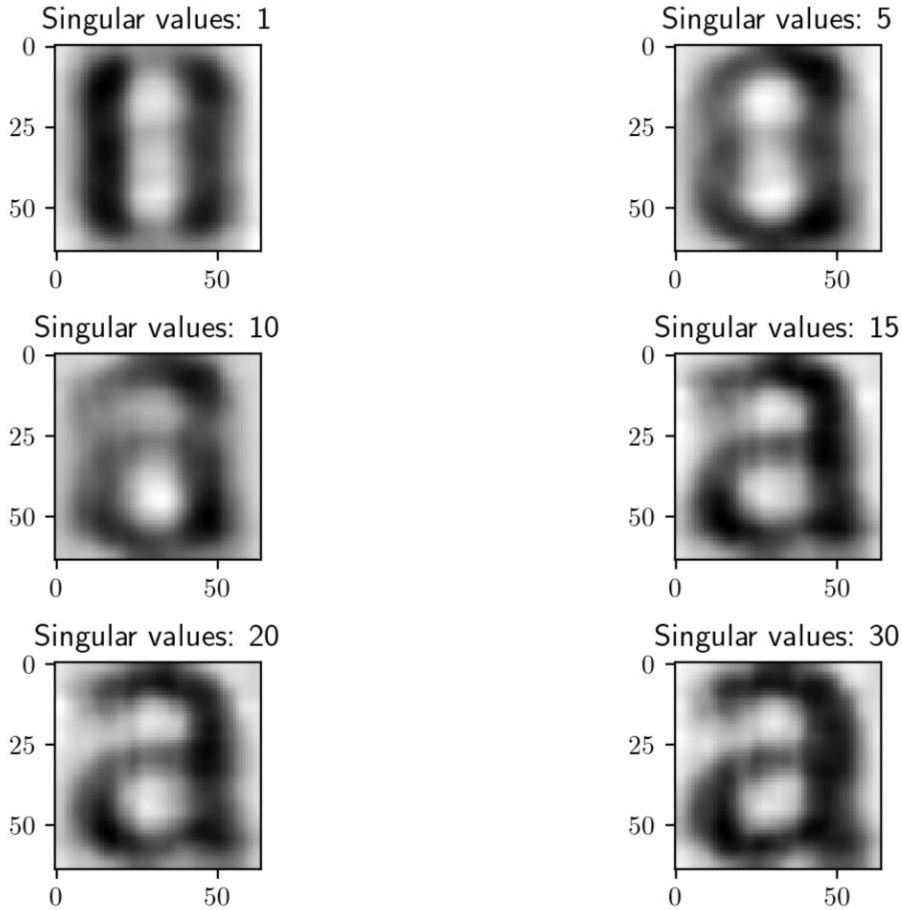
The resulting 12 eigen images are:



The graph of the first 10 components for the first 4 images is:



Finally, the resulting approximation for the letter “a” is:



The original image is (directly from the dataset):

Image Classification

For the classification with eigenvectors:

Character	Classification
d	a
j	y
l	i
n	v
p	e
q	a
u	a
y	v

Using diagonal elements of R_k:

Character	Classification
i	l
y	v

Using average R_k:

Character	Classification
g	q
y	v

Using diagonal elements of the average:

Character	Classification
f	t
y	v

Using identity:

Character	Classification
f	t
g	q
y	v

Methods 2, 3, and 4 have similar performance with least errors.

The tradeoff is between the accuracy of the data model and the accuracy that we get after estimating. The more complex model performs poorer when it comes to inference accuracy.

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from training_data.read_data import read_data, display_samples, datachar, read_data_test
import seaborn as sns
import pandas as pd

plt.rcParams['text.usetex'] = True

def excersise_2(p = 2, n = 1000):
    Rx = np.array([[2, -1.2],
                   [-1.2, 1]])
    # This is essentially equivalent to generating a
    # table of of gausiians each vith variance one in
    # a p X n matrix
    W = np.random.normal(0, 1, size = (p, n))
    eigenvalues, eigenvectors = np.linalg.eig(Rx)
    Lambd = np.sqrt(np.diag(eigenvalues))
    X_tild = Lambd@W
    X = eigenvectors@X_tild

    # Plot data
    def make_plot(A, name):
        plt.plot(A[0, :], A[1, :], '.')
        plt.title(name)
        plt.axis('equal')
        plt.show()

    make_plot(W, r'$W$ - white noise')
    make_plot(X_tild, r'$\tilde{X}$ -- decorrelated noise')
    make_plot(X, r'$X$ -- final noise')

    Z = X - np.average(X, axis=1)[:, np.newaxis]
    R_hat = 1/(n-1) * Z@Z.T
    print(f"Original: {Rx}")
    print(f"Estimate: {R_hat}")

    # Compute whitening opearion:
    eigenvalues, eigenvectors = np.linalg.eig(R_hat)
    X_hat_tilda = eigenvectors.T @ X
```

```
    W_est = np.diag(np.power(eigenvalues, -1/2))@X_hat_tilda
    make_plot(W_est, r"W -- whitened noise")
    make_plot(X_tild,  r'$\tilde{X}$ -- decorrelated noise')

    Z_W = W - np.average(W, axis=1)[:, np.newaxis]
    R_w = 1/(n-1) * Z_W@Z_W.T
    print(f"Estimated Rw: {R_w}")


def excersise_4():
    X = read_data()
    print(f"Shape of X: {X.shape}")
    dim, n = X.shape
    mean_image = np.average(X, axis=1)[:, np.newaxis]
    X = X - mean_image
    Z = X / np.sqrt(n-1)
    U, S, Vt = np.linalg.svd(Z, full_matrices=False)
    eigenvectors = U
    eigenvalues = S

    GY, GX = 3, 4
    top12eignevectors = eigenvectors[:, :12].T.reshape(12, 64, 64)
    _, ax = plt.subplots(3,4, constrained_layout = True)
    for i, eig in enumerate(top12eignevectors):
        ax[i//GX,i%4].imshow(eig,cmap=plt.cm.gray, interpolation='none')
        ax[i//GX,i%4].set_title(f"Eigenvector {i}")
    plt.show()
    assert np.all(eigenvalues[:-1] >= eigenvalues[1:]) # elements sorted
    Y = U.T @ X
    print(f"Shape of Y: {Y.shape}")
    print(f"Shape of U: {U.shape}")

    num_img = 4
    charaters = [datachar[i] for i in  range(num_img)]
    df = pd.DataFrame(Y[:10, :num_img], columns=charaters)
    df.reset_index(inplace=True)
    df_long = pd.melt(df, id_vars='index', value_vars=charaters,
                var_name='Character', value_name='Value')
    print(df_long)
    sns.lineplot(data=df_long, x='index', y='Value', hue='Character')
    plt.show()
```

```python
    _, ax = plt.subplots(3,2, constrained_layout = True)
    for i, m in enumerate([1, 5, 10, 15, 20, 30]):
        reconstructed_imgs = U[:, :m]@Y[:m, :] + mean_image
        ax[i//2,i%2].imshow(reconstructed_imgs[:,0].reshape(64, 64),cmap=plt.cm.gray, interpolation='none')
        ax[i//2,i%2].set_title(f"Singular values: {m}" )

    plt.show()

def excersise_5():
    X = read_data()
    print(f"Shape of X: {X.shape}")
    dim, n = X.shape
    mean_image = np.average(X, axis=1)[:, np.newaxis]
    X = X - mean_image
    Z = X / np.sqrt(n-1)
    U, S, Vt = np.linalg.svd(Z, full_matrices=False)
    A = U[:, :10]

    Y = A.T @ X

    Ck=12
    params={}


    for k in range(26):
        mu = np.average(Y[:, k::26], axis=1)[:, np.newaxis]
        Z = Y[:, k::26] - mu
        assert Z.shape == (10, Ck)
        params[k] = {
            "mean": mu,
            "cov": Z@Z.T / (Ck-1)
        }

    X_test = read_data_test()
    X_test -= mean_image
    Y_test = A.T @ X_test
    print(f"Shape of Y_test: {Y_test.shape}")
    print(f"Shape of X_test: {X_test.shape}")


    #compute Rwc
    Rwc = np.zeros((10, 10))
```

```
    for value in params.values():
        Rwc += value["cov"]
    Rwc /= 26


    results = np.zeros((26, 26))# class, input
    for k in range(26):
        y_test_shift = Y_test - params[k]['mean']
        for test_idx in range(26):
            y_test_sample = y_test_shift[:, test_idx]
            # Bk = params[k]['cov']
            Bk = np.diag(np.diag(params[k]['cov']))
            Bk = Rwc
            # Bk = np.diag(np.diag(Rwc))
            # Bk = np.identity(10)


            results[k, test_idx] = y_test_sample.T@np.linalg.inv(Bk)@y_test_sample + np.log(np.linalg.det(Bk))
    results = np.argmin(results, axis = 0)

    for i, x in enumerate(results):
        if i!= x:
            print(f"{datachar[i]} <-> {datachar[x]}")


if __name__ == "__main__":
    '''soruce driver code'''
    # excersise_2()
    # excersise_4()
    excersise_5()
```