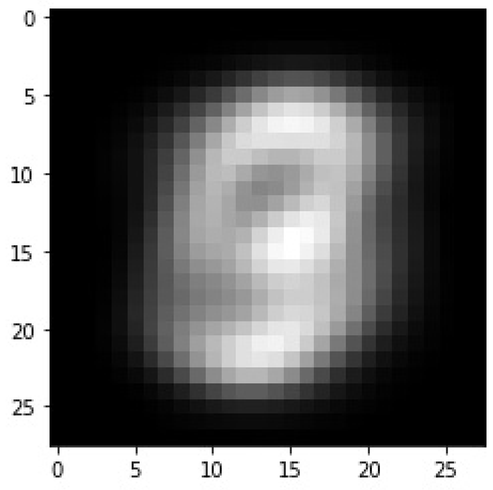
DSP phw1

November 2021

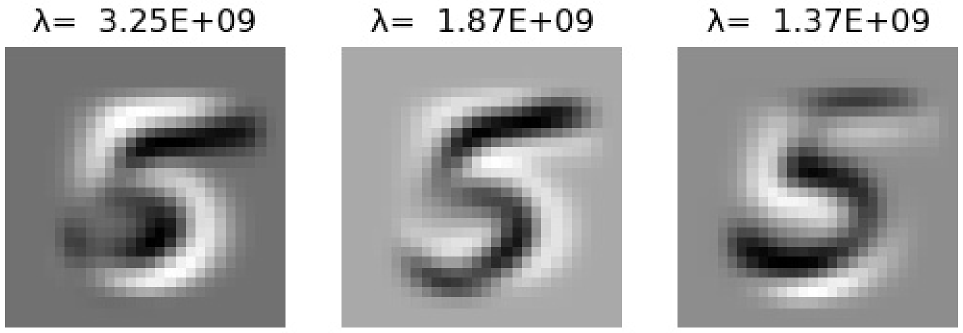
1 Q1

```
plt.imshow((mnist['data'].mean(axis=0)).reshape(28,28), 'gray')
```

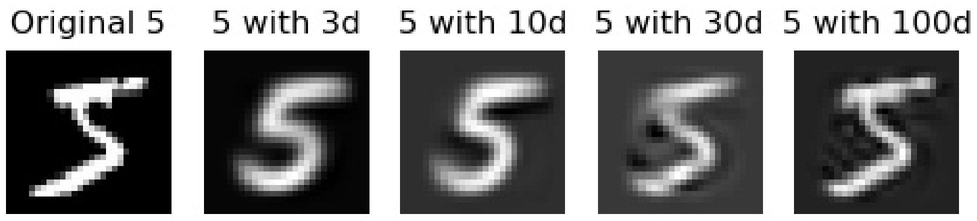
```
<matplotlib.image.AxesImage at 0x1d817ee3b20>
```



2 Q2

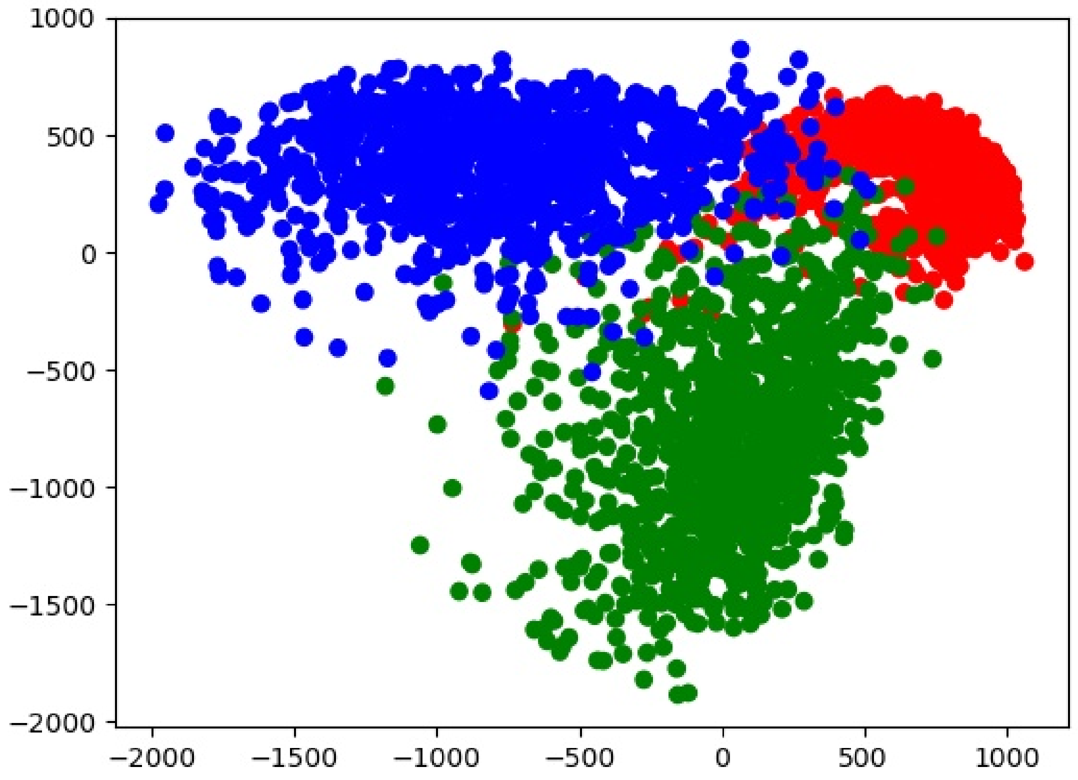


3 Q3



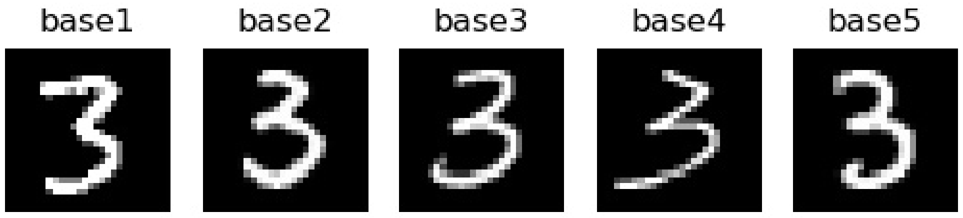
The image shows that the more eigenvectors are used to reconstruct the original image, the better it is reconstructed. The eigenvectors with smaller eigenvalues can refine some subtle features of the reconstructed image to make it look more similar to the original image.

4 Q4



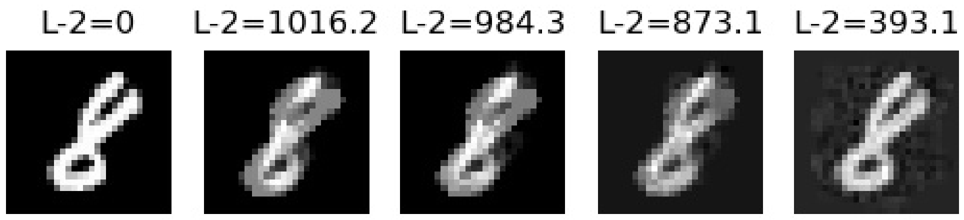
The "1"s are the red points, "3"s are green, and "6" are blue. The graph shows that same numbers tend to have similar coordinate when their dimension are reduced, this is because instances of a number often have similar features(in terms of the coefficients of eigenvectors).

5 Q5



As the image shows, all of the five bases are "3"s, this is because the #10001 image is also "3", and same numbers are likely to have large dot products.

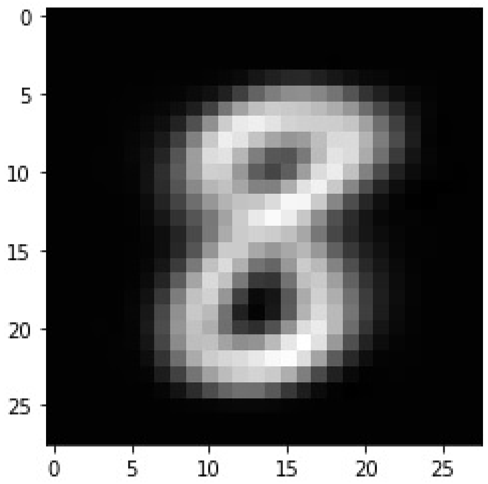
6 Q6



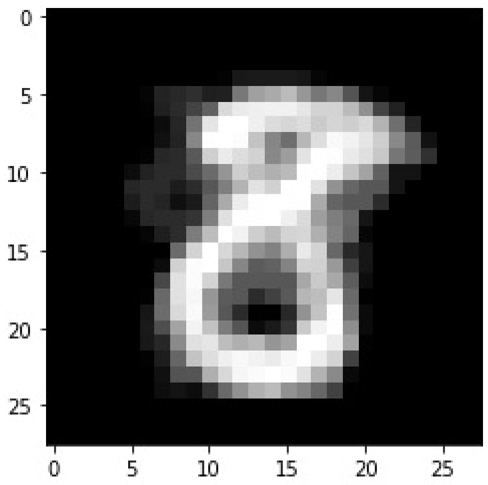
The larger sparsity applied, the better reconstructed image it can get, in terms of both appearances and L-2 errors. The reason is similar to Q3, more bases allows it to refine subtle features of the reconstructed image to make it look more similar to the original image.

7 Q7

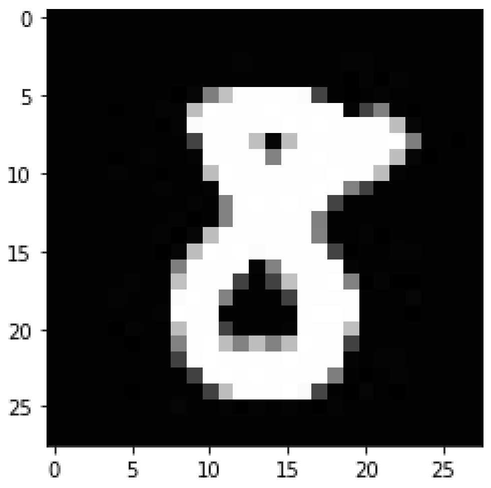
1.Use centered PCA to reconstruct the last "8". (Remain 5 largest eigenvalues.)



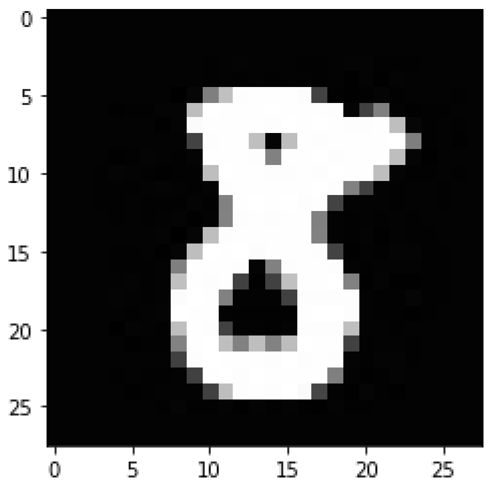
2.Use the first 6824 images as the base set. Use OMP to find the base and reconstruct the last "8". (Sparsity=5)

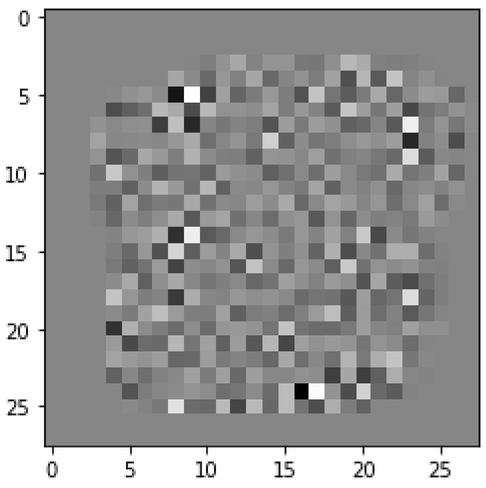


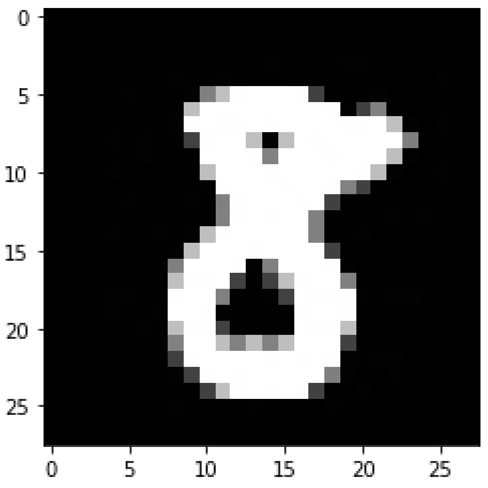
3.As 2, use "lasso" to find the bases and reconstruct the images.
The following image is reconstructed from lasso, fitted with alpha=1.

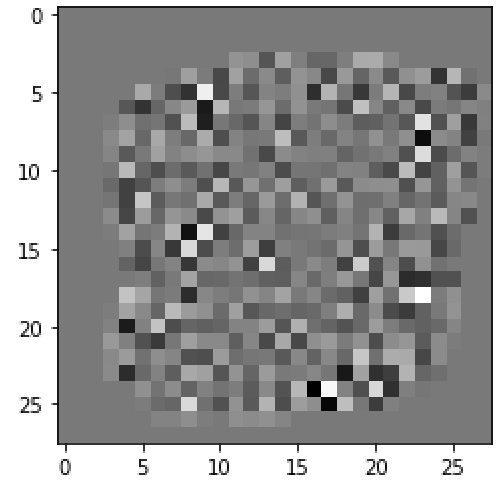


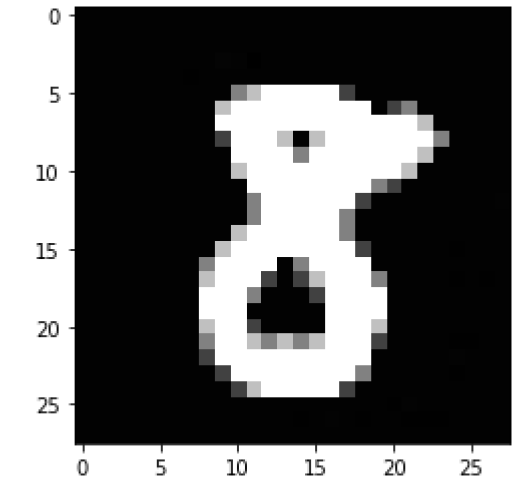
4.Adjust the lasso parameters. Explain your experiments and results.
The following images in order are:
1, lasso, alpha=1
2, The difference between (lasso, alpha=1) and the original image
3, lasso, alpha=0.5
4, The difference between (lasso, alpha=0.5) and the original image
5, lasso, alpha=0
6, The difference between (lasso, alpha=0) and the original image

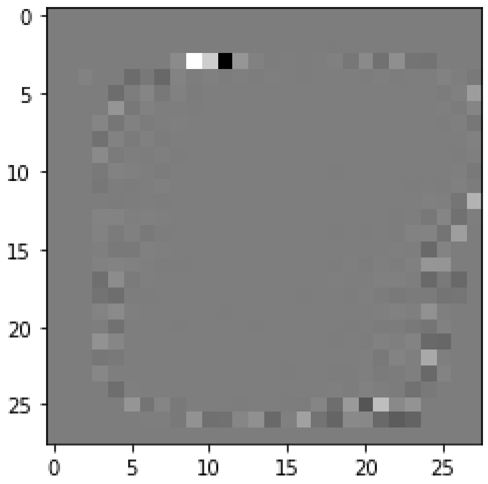



0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25

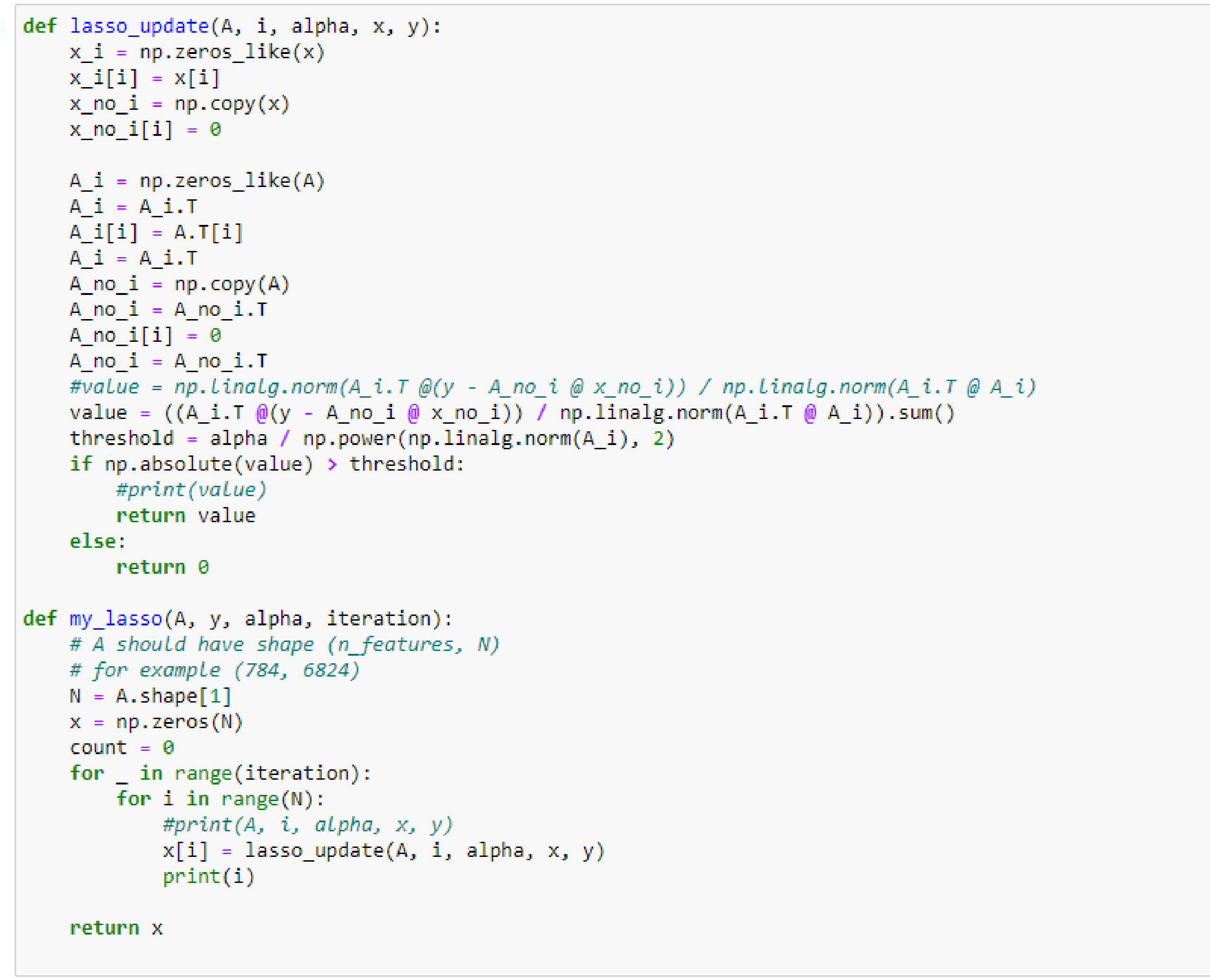

0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25



From the six image above, we see that in this case, the reconstructed images with different alpha have no big difference. However, we can still observe their difference by subtracting them with the original image. Also, the number of bases with non-zero coefficient are: (6824 with alpha=0), (1230 with alpha=0.5), and (920 with alpha=1), this is because larger alpha value implies a higher threshold for coefficients of the 6824 bases and they are more likely to be set to zero.

8 Bonus



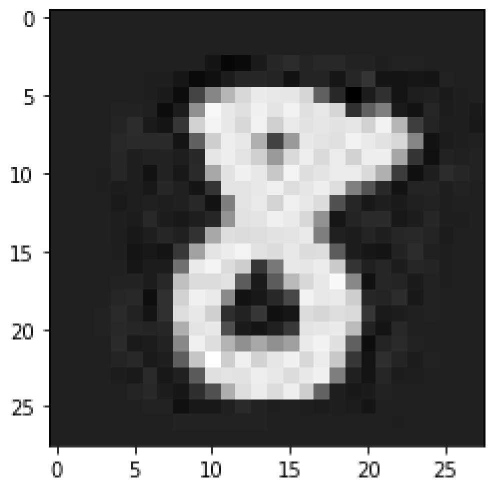
```python
def lasso_update(A, i, alpha, x, y):
    x_i = np.zeros_like(x)
    x_i[i] = x[i]
    x_no_i = np.copy(x)
    x_no_i[i] = 0

    A_i = np.zeros_like(A)
    A_i = A_i.T
    A_i[i] = A.T[i]
    A_i = A_i.T
    A_no_i = np.copy(A)
    A_no_i = A_no_i.T
    A_no_i[i] = 0
    A_no_i = A_no_i.T
    #value = np.linalg.norm(A_i.T @(y - A_no_i @ x_no_i)) / np.linalg.norm(A_i.T @ A_i)
    value = ((A_i.T @(y - A_no_i @ x_no_i)) / np.linalg.norm(A_i.T @ A_i)).sum()
    threshold = alpha / np.power(np.linalg.norm(A_i), 2)
    if np.absolute(value) > threshold:
        #print(value)
        return value
    else:
        return 0

def my_lasso(A, y, alpha, iteration):
    # A should have shape (n_features, N)
    # for example (784, 6824)
    N = A.shape[1]
    x = np.zeros(N)
    count = 0
    for _ in range(iteration):
        for i in range(N):
            #print(A, i, alpha, x, y)
            x[i] = lasso_update(A, i, alpha, x, y)
            print(i)

    return x
```

My implementation mainly follows ”Sparse representation L1-norm solutions.pdf” in Week 6, page 10. Row 16 corresponds to the last formula in the slide page, it is then passed through the thresholding function. Instead of repeat until convergence, user have to specify how many time the updating step should loop. Due to my lazy and poor implementation, it takes quite a lot of time for one iteration. The following image is the reconstructed image from my_lasso with one iteration and alpha=1.


0
5
10
15
20
25
0
5
10
15
20
25