Caitlyn Jones

Prof. Thomas Fai

MATH 122A: Numerical Methods and Big Data

17 November 2024

Homework 7

1. First, I generated matrix `x` with 5 randomly generated permuted coefficients and matrix `y` with random gaussian noise.

```
x = np.append((1+np.arange(5))/5,  np.zeros((1,128-5)))
x = np.random.permutation(x)

y = x + 0.05*np.random.normal(size=(1,128))
```

Below is the proof that the analytic solution for $\frac{1}{2}\|x-y\|^2+\lambda\frac{1}{2}\|x\|^2$ is $x=\frac{1}{x1+\lambda}y$.

$\frac{1}{2}\|x-y\|^2+\lambda\frac{1}{2}\|x\|^2$

$=\frac{1}{2}(x-y)^T(x-y)+\lambda\frac{1}{2}x^Tx$

$=\frac{1}{2}(x^Tx-2x^Ty+y^Ty)+\frac{1}{2}\lambda(x^Tx)$

$=\frac{1}{2}(x^Tx+\lambda x^Tx-2x^Ty+y^Ty)$

$=\frac{1}{2}((1+\lambda)x^Tx-2x^Ty+y^Ty)$

$\nabla_x=\frac{1}{2}((1+\lambda)(2x)-2y+0)$

$\nabla_x=(1+\lambda)(x)-y$

$(1+\lambda)x-y=0$

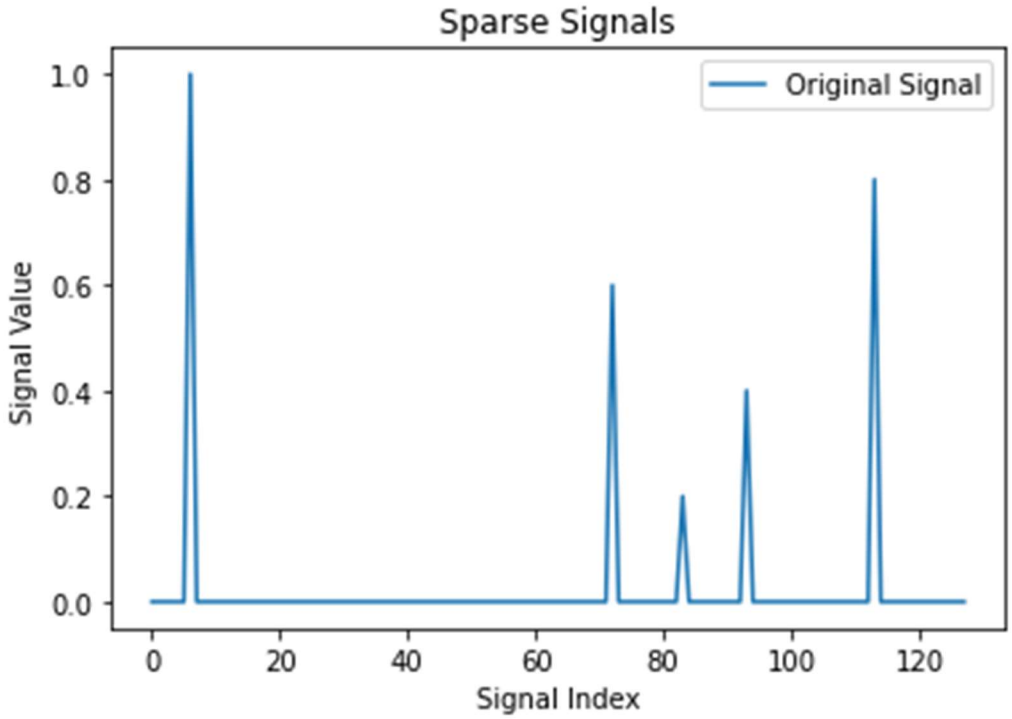
$(1+\lambda)x=y$

$x=\frac{y}{(1+\lambda)}$

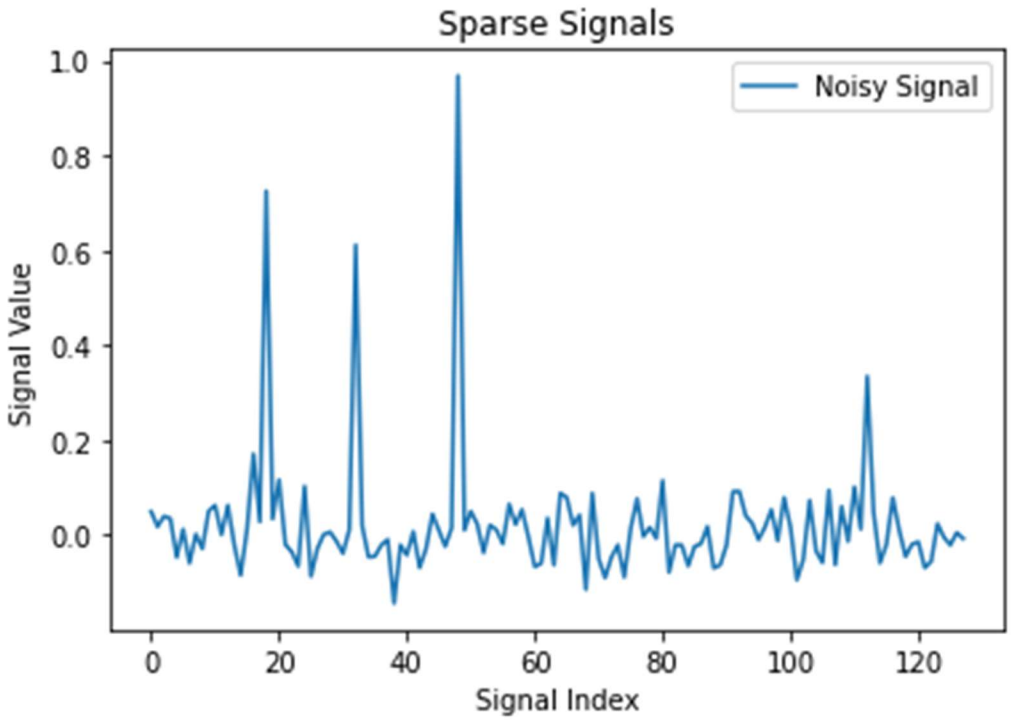
$x=\frac{1}{(1+\lambda)}y$ ■

In order to observe what happens with the given lambda values, I first graphed the original sparse signal.

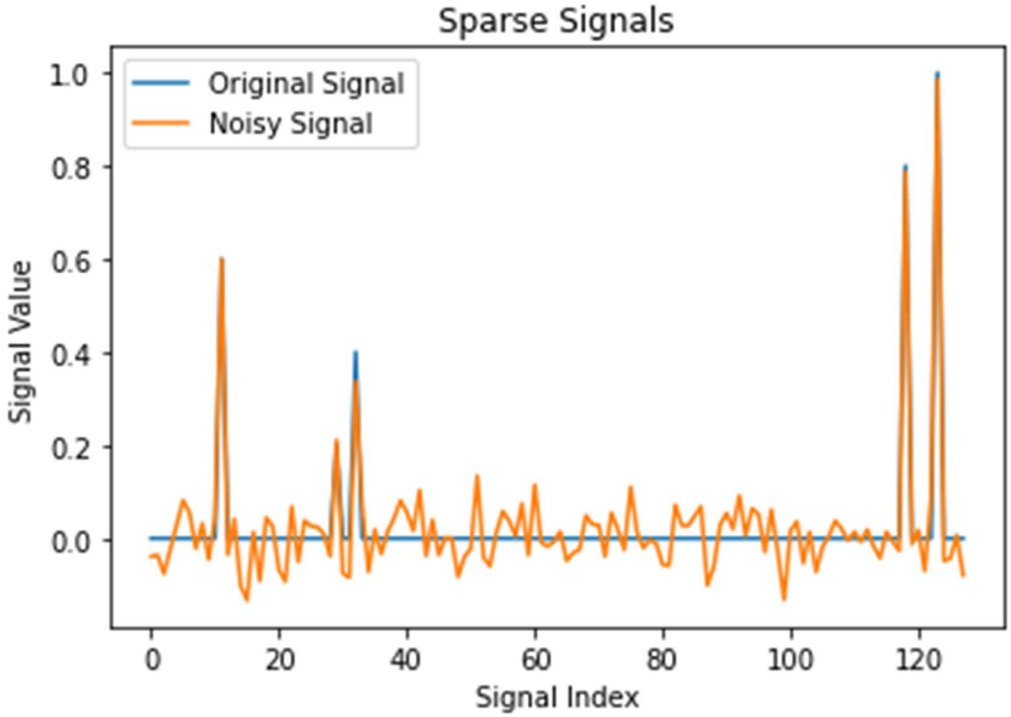
```
plt.figure()
plt.plot(x, label = "Original Signal")
#plt.plot(y[0], label = "Noisy Signal")
plt.legend()
plt.show()
```



Then, I graphed the sparse signal corrupted by the random noise.



And here are the original and noisy signals on the same graph.

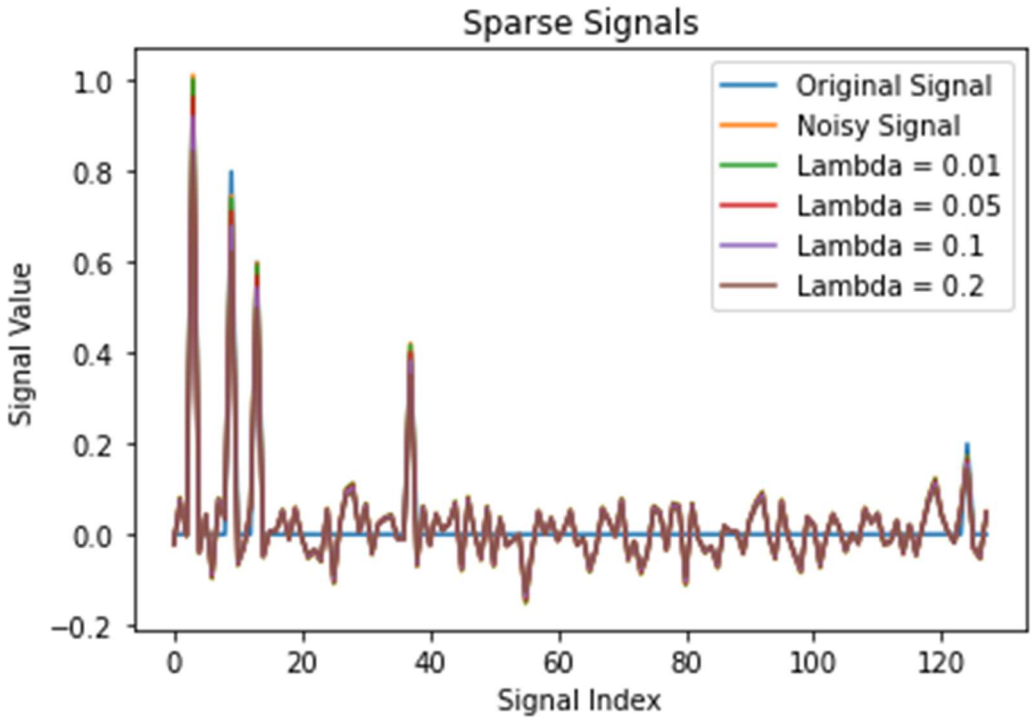


As you can see, the graphs have similar magnitudes, but the noisy graph has more peaks and divots due to the fact that we are adding noise at random throughout the graph.

From here, we will now be able to begin to denoise based on the various lambda values.

```
lambdas = [0.01, 0.05, 0.1, 0.2]
x_analytical = [(1/(1+lam))*y for lam in lambdas]
```

```
plt.figure()
plt.plot(x, label = "Original Signal")
plt.plot(y[0], label = "Noisy Signal")
plt.plot(x_analytical[0][0], label = "Lambda = 0.01")
plt.plot(x_analytical[1][0], label = "Lambda = 0.05")
plt.plot(x_analytical[2][0], label = "Lambda = 0.1")
plt.plot(x_analytical[3][0], label = "Lambda = 0.2")
plt.title("Sparse Signals")
plt.xlabel("Signal Index")
plt.ylabel("Signal Value")
plt.legend()
plt.show()
```

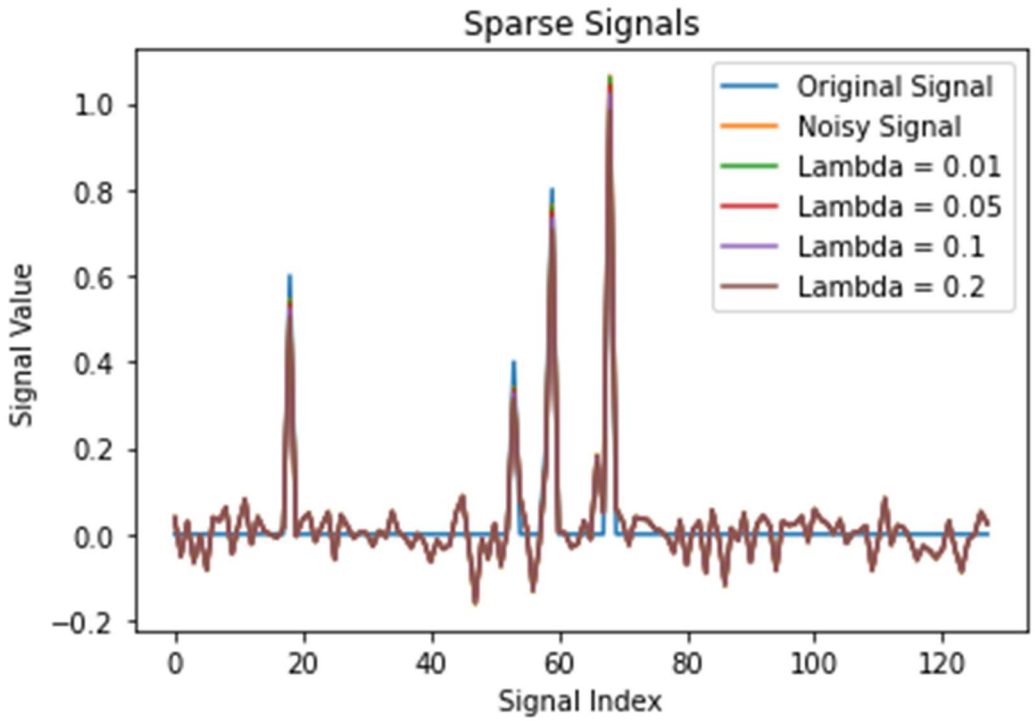


If you look really closely at the peaks and valleys of the graph, you can see how with each progressing lambda, the height of the graph becomes shorter. Now, we can compare these results with the python function.

```
ridge_solutions = [sklearn.linear_model.Ridge(alpha = lam,fit_intercept=False) for lam in lambdas]
x_ridge = [(r.fit(y.T, y.T).predict(y.T).flatten()) for r in ridge_solutions]
```

Here is that graph, which we can see is identical to the graph above using the analytic solution.

```
plt.figure()
plt.plot(x, label = "Original Signal")
plt.plot(y[0], label = "Noisy Signal")
plt.plot(x_ridge[0], label = "Lambda = 0.01")
plt.plot(x_ridge[1], label = "Lambda = 0.05")
plt.plot(x_ridge[2], label = "Lambda = 0.1")
plt.plot(x_ridge[3], label = "Lambda = 0.2")
plt.title("Sparse Signals")
plt.xlabel("Signal Index")
plt.ylabel("Signal Value")
plt.legend()
plt.show()
```



The solutions are sparse because they have many values close to zero, which we can see by the long sections small spikes close to zero. However, the solutions are interrupted by few nonzero values, which we can see by the four large spikes away from zero.

2. Below is the proof that the analytic solution for $\frac{1}{2}|x_i - y|^2 + \lambda|x_i|$ is

$$x = \begin{cases} y + \lambda, & if\ y < -\lambda \\ 0, & if\ |y| < \lambda \\ y - \lambda, & if\ y > \lambda \end{cases}$$

To start, we must find the gradient of the function and evaluate it based on the y condition in order to find the x value.

$\frac{1}{2}|x_i - y|^2 + \lambda|x_i|$

$\nabla_x = (x_i - y) + \lambda \frac{x_i}{|x_i|}$

When $y < -\lambda$, we can conclude that x will be negative.

So, $\nabla_x = (x_i - y) + \lambda \frac{x_i}{|x_i|}$ becomes $\nabla_x = (x_i - y) + \lambda(-1)$

Setting this equal to 0, we get the following:

$x_i - y - \lambda = 0$

$x_i = y + \lambda$ ■

Next, when $y > \lambda$, we know that x will be positive.

So, $\nabla_x = (x_i - y) + \lambda \frac{x_i}{|x_i|}$ becomes $\nabla_x = (x_i - y) + \lambda(1)$.

Setting this equation equal to 0, we get the following:

$x_i - y + \lambda = 0$

$x_i = y - \lambda$ ■

Lastly, if $x_i > 0$, then $\nabla_x = (x_i - y) + \lambda$ and if $x_i < 0$, then $\nabla_x = (x_i - y) - \lambda$.

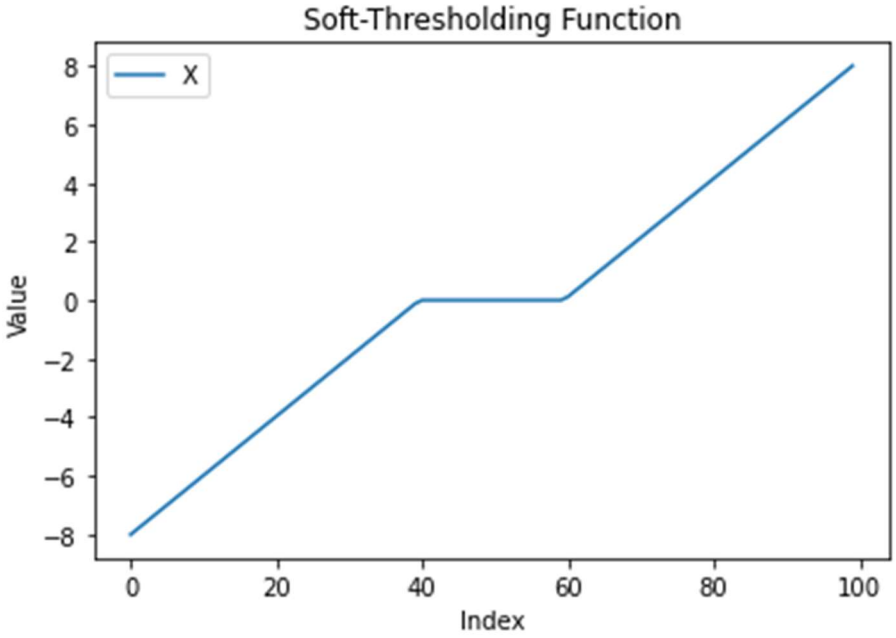
So, at $x_i = 0$, the optimal condition is $-\lambda \leq y \leq \lambda$, which means when $|y| < \lambda$, the minimizer is $x_i = 0$. ■

Next, I wrote a function that accepts y and λ and returns x.

```
def shrinkage(y, lam):
    if(y < -lam):
        x = y + lam
    if(y > lam):
        x = y - lam
    if(abs(y) < lam):
        x = 0
    return x
```

Then, I plotted the values for $y \in [-10,10]$ and $\lambda = 2$.

```
y = np.linspace(-10,10,100)
lam = 2
x = [shrinkage(i,lam) for i in y]
```



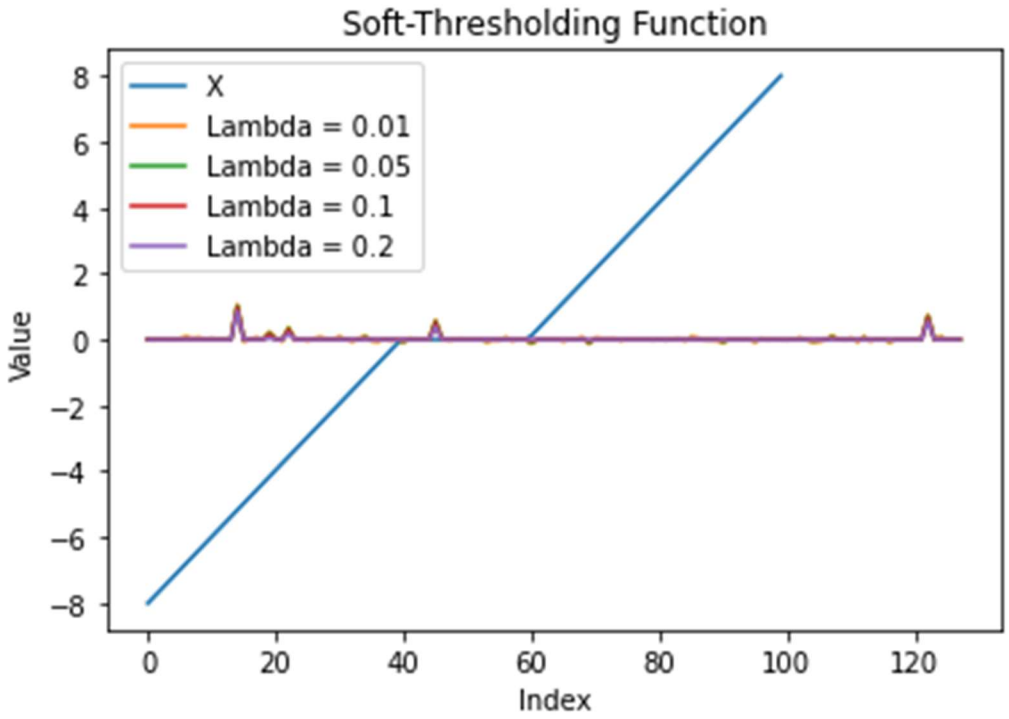
When y is small, the values are shrinking closer to zero from -10 and when y is big, the values are shrinking closer to 10. Next, I applied the function to the noisy signal with the various lambda values.

```
Nsamples = 128

no_noise = np.append((1+np.arange(5))/5,  np.zeros((1,128-5)))
no_noise = np.random.permutation(no_noise)
noisy = no_noise + 0.05*np.random.normal(size=Nsamples)

lambdas = [0.01, 0.05, 0.1, 0.2]
x_noisy = {lamb:[shrinkage(n,lamb) for n in noisy] for lamb in lambdas }
```

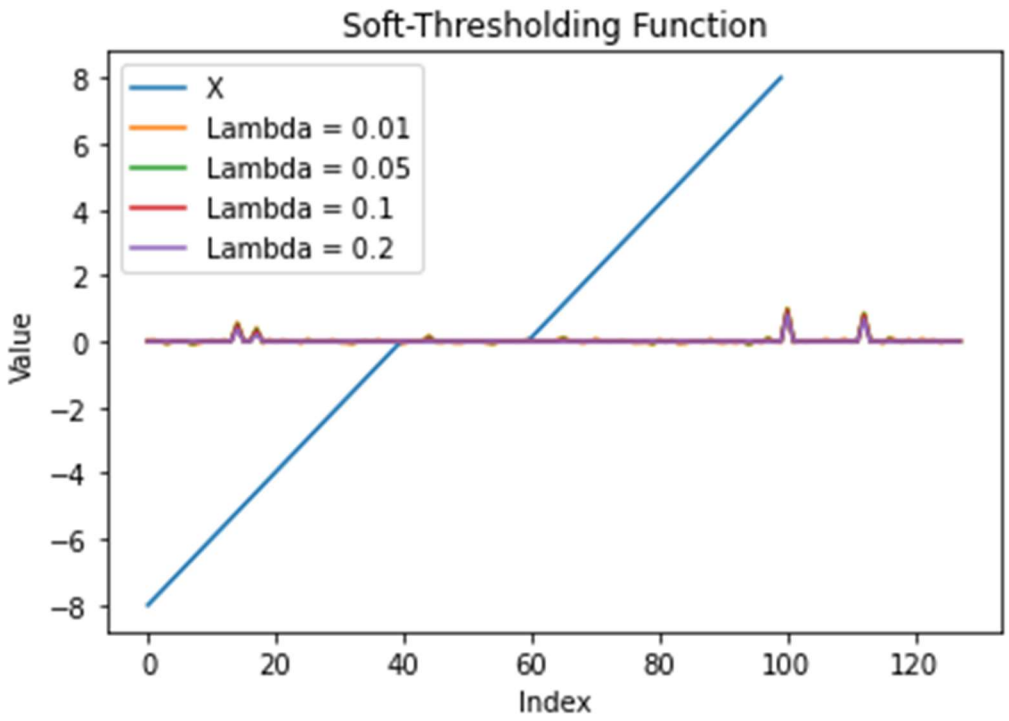
Next, I graphed the values for `x_noisy`.



Then, I verified my results by using the Python function, which will produce a very similar graph to the one above.

```
lasso_solutions = {}
for l in lambdas:
    alpha = l / Nsamples
    lasso = sklearn.linear_model.Lasso(alpha=alpha, fit_intercept=False)
    lasso.fit(np.eye(Nsamples), noisy)
    lasso_solutions[l] = lasso.coef_

plt.figure()
plt.plot(x, label = "X")
plt.plot(lasso_solutions[0.01], label = "Lambda = 0.01")
plt.plot(lasso_solutions[0.05], label = "Lambda = 0.05")
plt.plot(lasso_solutions[0.1], label = "Lambda = 0.1")
plt.plot(lasso_solutions[0.2], label = "Lambda = 0.2")
plt.title("Soft-Thresholding Function")
plt.xlabel("Index")
plt.ylabel("Value")
plt.legend()
plt.show()
```



The solutions are sparse, like before, because they are all very close to zero. Unlike in problem 1, we can clearly see the sparsity of these solutions because they are aligned on the values of x that equal 0.

3.

```
file = "ratings-train.csv"
data = pd.read_csv(file)
users = np.array(data.iloc[:, 0])
movies = np.array(data.iloc[:, 1])
movie_ratings = np.array(data.iloc[:, 2])

test_file = "ratings-test.csv"
test_data = pd.read_csv(test_file)
test_users = np.array(test_data.iloc[:, 0])
test_movies = np.array(test_data.iloc[:, 1])
test_movie_ratings = np.array(test_data.iloc[:, 2])

num_users = users.max() + 1
num_movies = movies.max() + 1

n = 10
X = np.random.rand(num_users,n)
Y = np.random.rand(num_movies,n)

reg = 0.1
iterations = 20

for i in range(iterations):
    for u in range(num_users):
        rated_movies = movies[users == u]
        ratings = movie_ratings[users == u]

        if len(rated_movies) > 0:
            Y_rated = Y[rated_movies]
            X[u] = np.linalg.solve(Y_rated.T @ Y_rated + reg * np.eye(n), Y_rated.T @ ratings)

    for m in range(num_movies):
        rated_users = users[movies == m]
        ratings = movie_ratings[movies == m]

        if len(rated_users) > 0:
            X_rated = X[rated_users]
            Y[m] = np.linalg.solve(X_rated.T @ X_rated + reg * np.eye(n), X_rated.T @ ratings)

def predict_rating(user_id, movie_id):
    return X[user_id] @ Y[movie_id].T

print(predict_rating(test_users[0][0],test_movies[0][0]))
```