Activity 17 ECE 532.

Ayan Deep Hazra

1.

a) Given, $A^T A A^T + \lambda A^T$

$$A^T(AA^T + \lambda I) = (A^TA + \lambda I)A^T$$

$$(A^TA+\lambda I)^{-1}A^T\cancel{(AA^T+\lambda I)(AA^T+\lambda I)^{-1}} =$$

$$\cancel{(A^TA)+\lambda I)^{-1}(A^TA+\lambda I)}\,A^T(AA^T+\lambda I)^{-1}$$

$\therefore$, $(A^TA + \lambda I)^{-1}A^T = A^T(AA^T + \lambda I)^{-1}$

b) Since $A \in \mathbb{R}^{5000 \times 100}$

we have $AA^T \in \mathbb{R}^{5000 \times 5000}$

and $A^TA \in \mathbb{R}^{100 \times 100}$

Thus the $(A^TA + \lambda I)^{-1}A^T$ ~~f~~formula will calculate inverse faster. This is because operating on a 100×100 matrix will be faster than doing the same operation of 5000×5000 matrix

c) i) $y_i = \text{sign}\{g_i^T w\}$

$y = 100 \times 1 \quad g = 8000 \times 100 \quad w = 8000 \times 1$

$\min_w \|g^T w - y\|_2^2 \Rightarrow w = (A^T A)^{-1} A^T y$

$= (g^T g) g^T y$

As the number of columns outweighs the number of rows, we conclude that the system has no unique solutions due to no linearly independent columns.

ii) $g^T g = 100 \times 100$ as stated.

Thus for Tikhonov we have

$\min_w \|g^T w + \lambda I - y\|_2^2$

$\Rightarrow w = (g^T g + \lambda I)^{-1} g^T y$

solution.

$(A^T A + \lambda I)^{-1} A^T y$ form is more computationally efficient as $g^T g$ is 100×100

2\. a) Given, $\min_{w} \|z - w\|_2^2 + \lambda \|w\|_2^2$

$$= \min_{w} \left[\sum_{i=1}^{n} (z_i - w_i)^2 + \lambda w_i^2 \right]$$

Clearly the problem is seperable as the ith term does not depend on any other i-index terms.

$$= \sum \min_{w_i} \left[(z_i - w_i)^2 + \lambda w_i^2 \right]$$

$$= \min_{w_1} \left[(z_1 - w_1)^2 + \lambda w_1^2 \right] + \min_{w_2} \left[(z_2 - w_2)^2 + \lambda w_2^2 \right]$$

$$+ \dots\dots + \min_{w_n} \left[(z_n - w_n)^2 + \lambda w_n^2 \right]$$

b) Given, $\min_{w} \|z - w\|_2^2 + \lambda \|w\|_1$

$$= \min_{w} \sum_{i=1}^{n} (z_i - w_i)^2 + \lambda |w_i|$$

Clearly the problem is seperable as the the i^{th} term does not depend on any other i-index terms.

$$= \sum \min_{w_i} \left[(z_i - w_i)^2 + \lambda |w_i| \right]$$

$$= \min_{w_1} \left[(z_1 - w_1)^2 + \lambda |w_1| \right] + \min_{w_2} \left[(z_2 - w_2)^2 + \lambda |w_2| \right]$$

$$+ \dots\dots + \min_{w_3} \left[(z_3 - w_3)^2 + \lambda |w_3| \right]$$

Activity 17

Setup

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
def prxgraddescent_l2(X,y,tau,lam,w_init,it):

## compute it iterations of L2 proximal gradient descent starting at w1
## w_{k+1}= (w_k - tau*X'*(X*w_k - y)/(1+lam*tau)
## step size tau
    W = np.zeros((w_init.shape[0], it+1))
    Z = np.zeros((w_init.shape[0], it+1))
    W[:,[0]] = w_init
    for k in range(it):
        Z[:,[k+1]] = W[:,[k]] - tau * X.T @ (X @ W[:,[k]] - y);
        W[:,[k+1]] = Z[:,[k+1]]/(1+lam*tau)

    return W,Z
```

In [3]:
```
## Proximal gradient descent trajectories
## Least Squares Problem
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 0.5]])
Sinv = np.linalg.inv(S)
V = 1/np.sqrt(2)*np.array([[1, 1], [1, -1]])
y = np.array([[np.sqrt(2)], [0], [1], [0]])

X = U @ S @ V.T

### Find Least Squares Solution
w_ls = V @ Sinv @ U.T @ y
c = y.T @ y - y.T @ X @ w_ls

### Find values of f(w), the contour plot surface for
w1 = np.arange(-1,3,.1)
w2 = np.arange(-1,3,.1)
fw = np.zeros((len(w1), len(w2)))
for i in range(len(w2)):
    for j in range(len(w1)):
        w = np.array([ [w1[j]], [w2[i]] ])
        fw[i,j] = (w-w_ls).T @ X.T @ X @ (w-w_ls) + c
```

Question 3a)

In [4]:

```
## Find and display weights generated by gradient descent

w_init = np.array([[-1],[1]])
lam = 0.5;
it = 20
tau = 0.5
W,Z = prxgraddescent_l2(X,y,tau,lam,w_init,it)

maxValueTau = 1/(np.linalg.norm(X, ord=2))**2

print("maximum value for the step size τ that will guarantee convergence: ", maxValueTau)
```

```
maximum value for the step size τ that will guarantee convergence:  0.9999999999999996
```

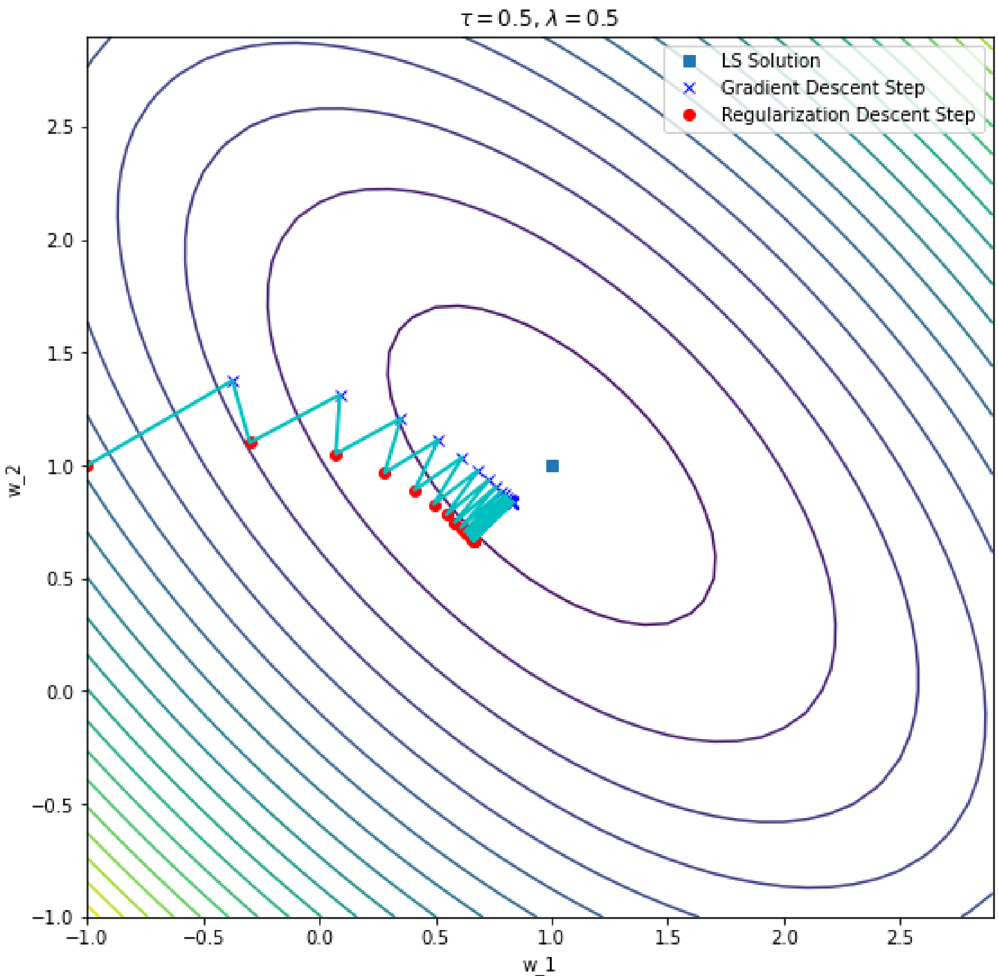
Question 3b)

In [5]:

```


# Concatenate gradient and regularization steps to display trajectory
G = np.zeros((2,0))
for i in range(it):
    G = np.hstack((G,np.hstack((W[:,[i]],Z[:,[i+1]]))))

plt.figure(figsize=(9,9))
plt.contour(w1,w2,fw,20)
plt.plot(w_ls[0],w_ls[1],"s", label="LS Solution")
plt.plot(Z[0,1::],Z[1,1:],'bx',linewidth=2, label="Gradient Descent Step")
plt.plot(W[0,:],W[1,:],'ro',linewidth=2, label="Regularization Descent Step")
plt.plot(G[0,:],G[1,:],'-c',linewidth=2)
plt.legend()
plt.xlabel('w_1')
plt.ylabel('w_2')
plt.title('$\\tau = $'+str(.5)+', $\lambda = $'+str(lam));
```



Ridge regression minimizes the norm of w, which pushes our w towards the origin (optimum weights). Gradient descent then uses the contour at that point to make the next step, within the limits of tau. The points alternate between going towards the origin point and following the negative gradient of the contour.

Question 3c)

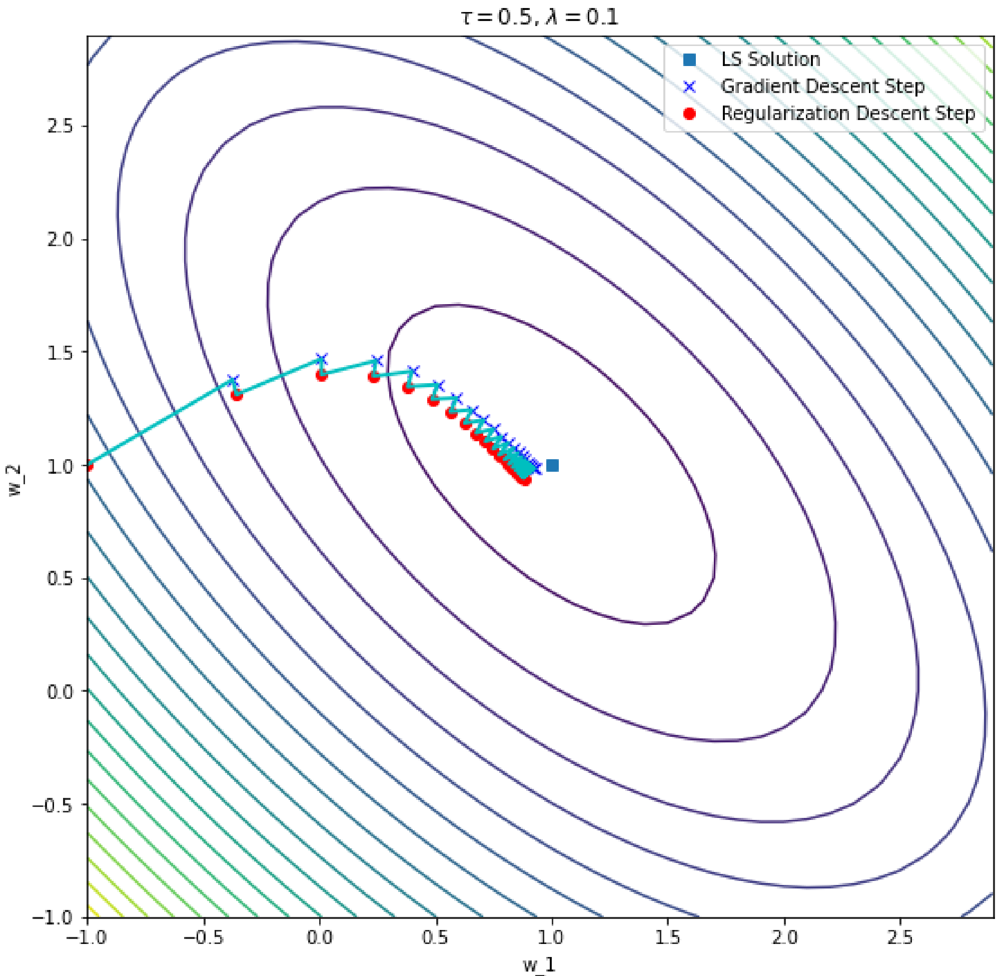
In [6]:

```
## Find and display weights generated by gradient descent

w_init = np.array([[-1],[1]])
lam = 0.1;
it = 20
tau = 0.5
W,Z = prxgraddescent_l2(X,y,tau,lam,w_init,it)

# Concatenate gradient and regularization steps to display trajectory
G = np.zeros((2,0))
for i in range(it):
    G = np.hstack((G,np.hstack((W[:,[i]],Z[:,[i+1]]))))

plt.figure(figsize=(9,9))
plt.contour(w1,w2,fw,20)
plt.plot(w_ls[0],w_ls[1],"s", label="LS Solution")
plt.plot(Z[0,1::],Z[1,1:],'bx',linewidth=2, label="Gradient Descent Step")
plt.plot(W[0,:],W[1,:],'ro',linewidth=2, label="Regularization Descent Step")
plt.plot(G[0,:],G[1,:],'-c',linewidth=2)
plt.legend()
plt.xlabel('w_1')
plt.ylabel('w_2')
plt.title('$\\tau = $'+str(.5)+', $\lambda = $'+str(lam));
```



We notice that unlike in the 3b case, the lambda parameter does not act as a good opposing force this time, having only minimal changes to each iteration for the descent as compared to the ridge regression. Thus the trajectory appears to be more accurate.