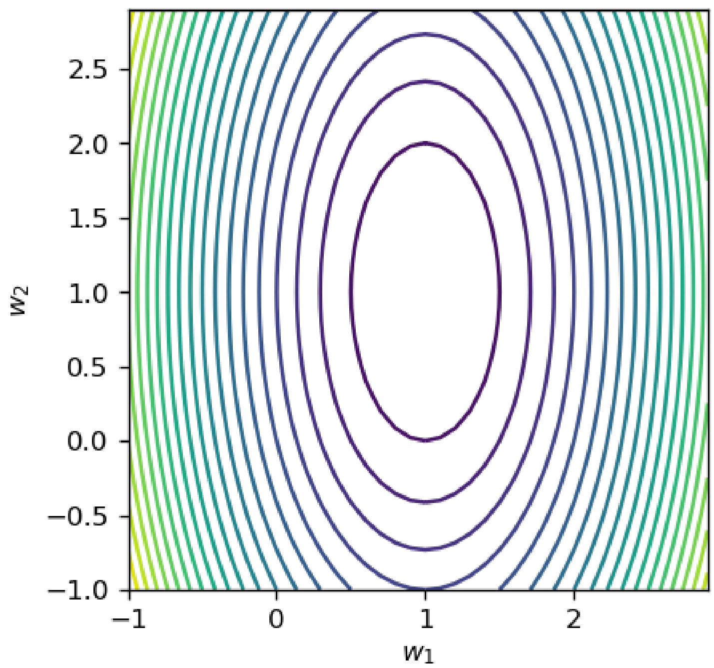
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
## DO NOT Change
def graddescent(X,y,tau,w_init,it):
    """
    compute 10 iterations of gradient descent starting at w1
    w_{k+1}= w_k - tau*X'*(X*w_k - y)
    """
    W = np.zeros((w_init.shape[0],it))
    W[:,[0]] = w_init
    for k in range(it-1):
        W[:,[k+1]] = W[:,[k]] - tau * X.T @ (X @ W[:,[k]] - y)
    return W
```

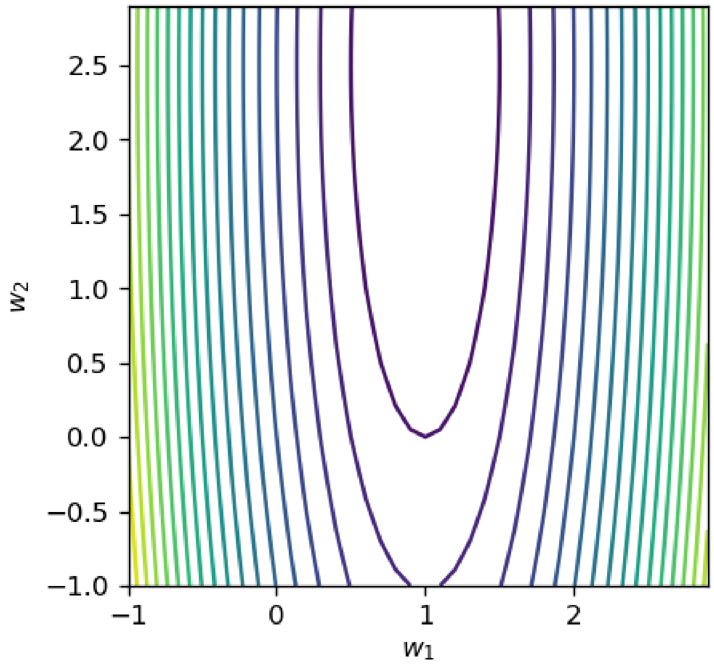
Question 1b)

In [3]:
```
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 = np.eye(2)
X = U @ S @ V.T
y = np.array([[1], [0.5], [1], [0]])

### 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

### Plot the countours
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.xlim([-1,3])
plt.ylim([-1,3])
plt.xlabel('$w_1$')
plt.ylabel('$w_2$')
plt.axis('square');
```



Question 1c)

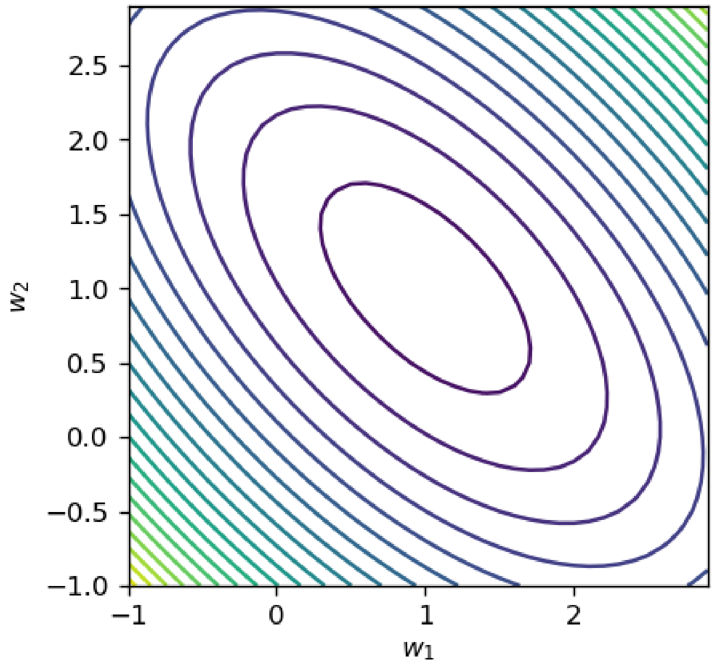
In [4]:
```
## Copy and paste code from 1b
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 0.2]])
Sinv = np.linalg.inv(S)
V = np.eye(2)
X = U @ S @ V.T
y = np.array([[1], [0.5], [1], [0]])

### 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

### Plot the countours
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.xlim([-1,3])
plt.ylim([-1,3])
plt.xlabel('$w_1$')
plt.ylabel('$w_2$')
plt.axis('square');
```



It makes the contour's ellipses more eccentric that is the gradient along w2 decreases (the slope becomes less intense).

Question 1d)

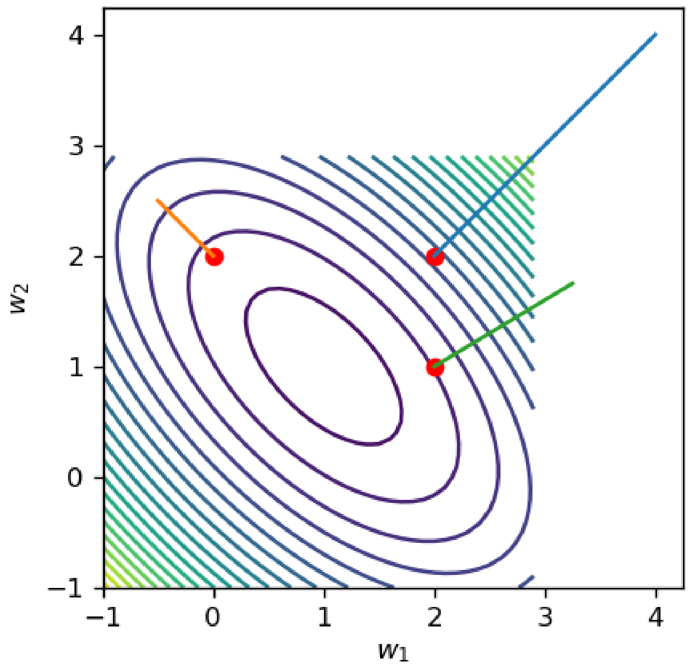
In [5]:
```
## Copy and paste code from 1b
## Copy and paste code from 1b
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 0.5]])
Sinv = np.linalg.inv(S)
V = np.array([[2**-0.5,2**-0.5], [2**-0.5, -1*(2**-0.5)]])
X = U @ S @ V.T
y = np.array([[2**0.5], [0], [1], [0]])

### 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

### Plot the countours
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.xlim([-1,3])
plt.ylim([-1,3])
plt.xlabel('$w_1$')
plt.ylabel('$w_2$')
plt.axis('square');
```



It changes the orientation of the eliiptical contours from being arranged along wi, w2 axes to axes that are at an angle to w1, w2.

Question 1e)

In [6]:
```
wi = np.array([[2],[2]])
wii = np.array([[0],[2]])
wiii = np.array([[2],[1]])

gradi = 2* X.T @ (X@wi-y)
gradii = 2*X.T @ (X@wii-y)
gradiii = 2*X.T @ (X@wiii-y)

### Plot the countours
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.plot(wi[0,0], wi[1,0], 'ro')
plt.plot([wi[0,0],wi[0,0]+gradi[0,0]], [wi[1,0],wi[1,0]+gradi[1,0]])
plt.plot(wii[0,0], wii[1,0], 'ro')
plt.plot([wii[0,0],wii[0,0]+gradii[0,0]], [wii[1,0],wii[1,0]+gradii[1,0]])
plt.plot(wiii[0,0], wiii[1,0], 'ro')
plt.plot([wiii[0,0],wiii[0,0]+gradiii[0,0]], [wiii[1,0],wiii[1,0]+gradiii[1,0]])
plt.xlim([-1,3])
plt.ylim([-1,3])
plt.xlabel('$w_1$')
plt.ylabel('$w_2$')
plt.axis('square');
```



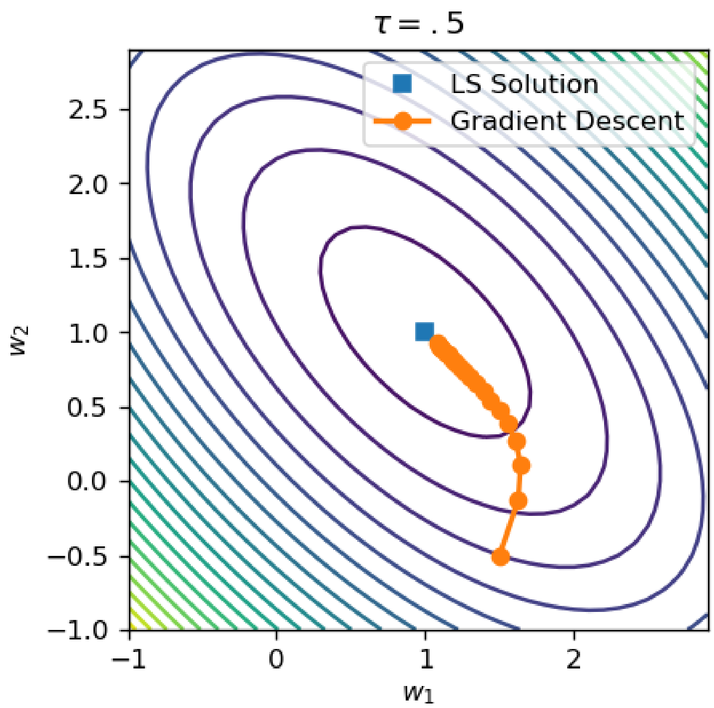
Question 2a)

The maximum value of Tau that will guarantee convergence is 2.

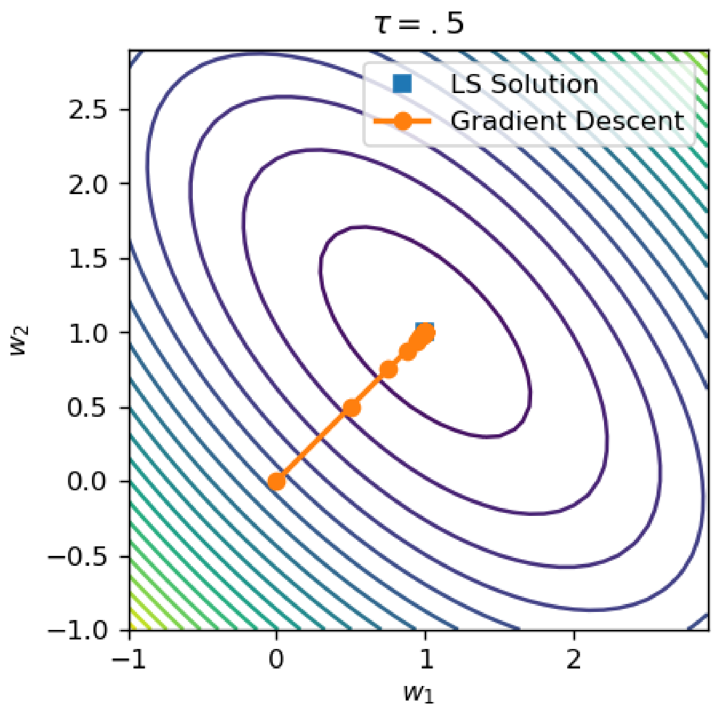
Question 2b)

In [7]:
```
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]])
X = U @ S @ V.T

y = np.array([[np.sqrt(2)], [0], [1], [0]])

### 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(w1)):
    for j in range(len(w2)):
        w = np.array([ [w1[i]], [w2[j]] ])
        fw[i,j] = (w-w_ls).T @ X.T @ X @ (w-w_ls) + c
```

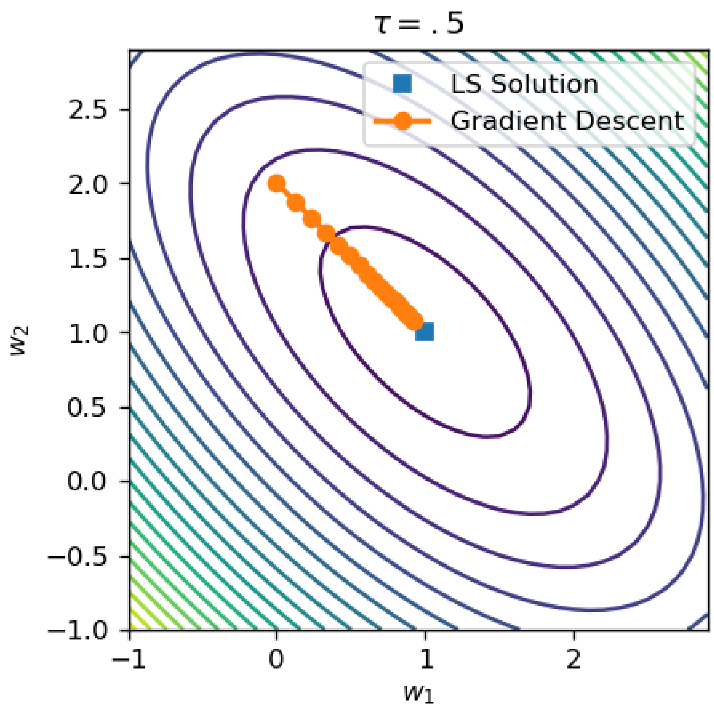
In [8]:
```
w_init = np.array([[1.5],[-0.5]])  # complete this line with a 2x1 numpy array for the values specified in the ac
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);


### Create plot
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.plot(w_ls[0],w_ls[1],"s", label="LS Solution")
plt.plot(W[0,:],W[1,:],'o-',linewidth=2, label="Gradient Descent")
plt.legend()
plt.xlim([-1,3])
plt.xlabel('$w_1$')
plt.ylim([-1,3])
plt.ylabel('$w_2$')
plt.title(r'$\tau = .5$');
plt.axis('square');
```



In [9]:
```
w_init = np.array([[0],[0]])  # complete this line with a 2x1 numpy array for the values specified in the activit
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);


### Create plot
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.plot(w_ls[0],w_ls[1],"s", label="LS Solution")
plt.plot(W[0,:],W[1,:],'o-',linewidth=2, label="Gradient Descent")
plt.legend()
plt.xlim([-1,3])
plt.xlabel('$w_1$')
plt.ylim([-1,3])
plt.ylabel('$w_2$')
plt.title(r'$\tau = .5$');
plt.axis('square');
```

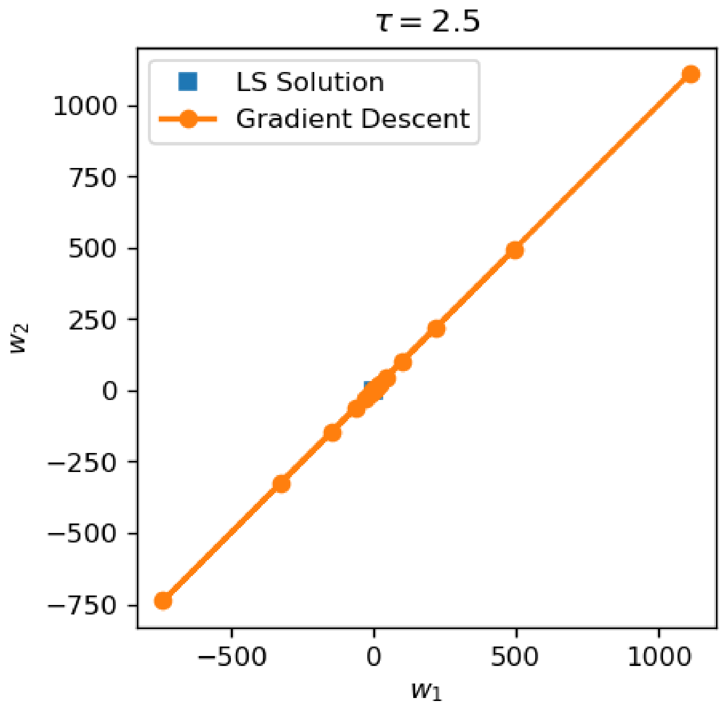


In [10]:
```
w_init = np.array([[0],[2]])  # complete this line with a 2x1 numpy array for the values specified in the activity
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);


### Create plot
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.plot(w_ls[0],w_ls[1],"s", label="LS Solution")
plt.plot(W[0,:],W[1,:],'o-',linewidth=2, label="Gradient Descent")
plt.legend()
plt.xlim([-1,3])
plt.xlabel('$w_1$')
plt.ylim([-1,3])
plt.ylabel('$w_2$')
plt.title(r'$\tau = .5$');
plt.axis('square');
```



Question 2c)

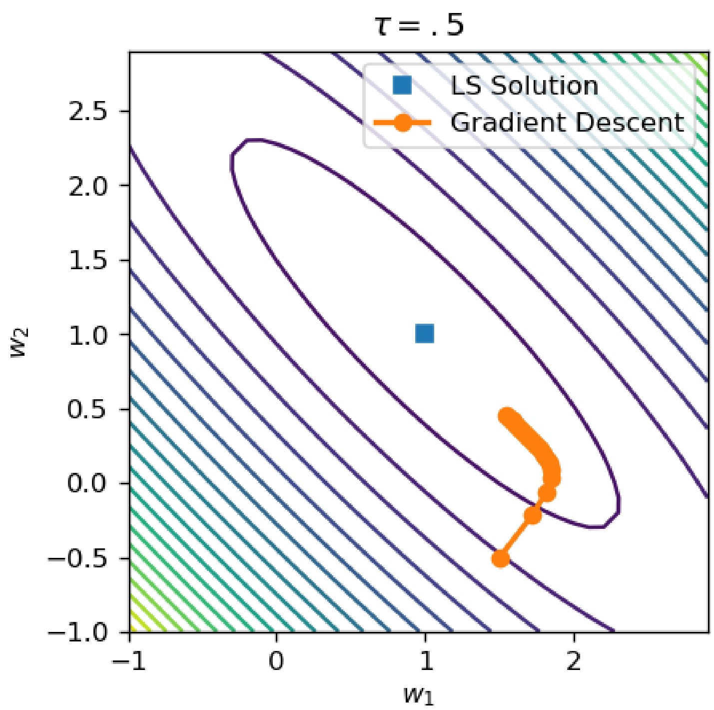
In [11]:
```
# copy and paste code from above
w_init = np.array([[1.5],[-0.5]])  # complete this line with a 2x1 numpy array for the values specified in the ac
it = 20
tau = 2.5
W = graddescent(X,y,tau,w_init,it);


### Create plot
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.plot(w_ls[0],w_ls[1],"s", label="LS Solution")
plt.plot(W[0,:],W[1,:],'o-',linewidth=2, label="Gradient Descent")
plt.legend()
plt.xlim([-1,3])
plt.xlabel('$w_1$')
plt.ylim([-1,3])
plt.ylabel('$w_2$')
plt.title(r'$\tau = 2.5$');
plt.axis('square');
```



Tau is larger than the maximum value which allows for converge. Thus we get a graph where is no convergence.

Question 2d)

In [12]:
```
## Copy and paste code from above
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 0.25]])
Sinv = np.linalg.inv(S)
V = 1/np.sqrt(2)*np.array([[1, 1], [1, -1]])
X = U @ S @ V.T
y = np.array([[np.sqrt(2)], [0], [1], [0]])

### 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(w1)):
    for j in range(len(w2)):
        w = np.array([ [w1[i]], [w2[j]] ])
        fw[i,j] = (w-w_ls).T @ X.T @ X @ (w-w_ls) + c
w_init = np.array([[1.5],[-0.5]])  # complete this line with a 2x1 numpy array for the values specified in the ac
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);


### Create plot
plt.figure(num=None, figsize=(4, 4), dpi=120)
plt.contour(w1,w2,fw,20)
plt.plot(w_ls[0],w_ls[1],"s", label="LS Solution")
plt.plot(W[0,:],W[1,:],'o-',linewidth=2, label="Gradient Descent")
plt.legend()
plt.xlim([-1,3])
plt.xlabel('$w_1$')
plt.ylim([-1,3])
plt.ylabel('$w_2$')
plt.title(r'$\tau = .5$');
plt.axis('square');
```



The step size we plot gets even smaller ad smaller, so if we assume changes to the singular value is really small, we make the contours flatter and which slows the trajectory and thus convergence will take really long.

Question 2e)

Small ratio of the singular values leads to more iterations to get to convergence, and therefore results in a flatter cost function.

In [ ]: