In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

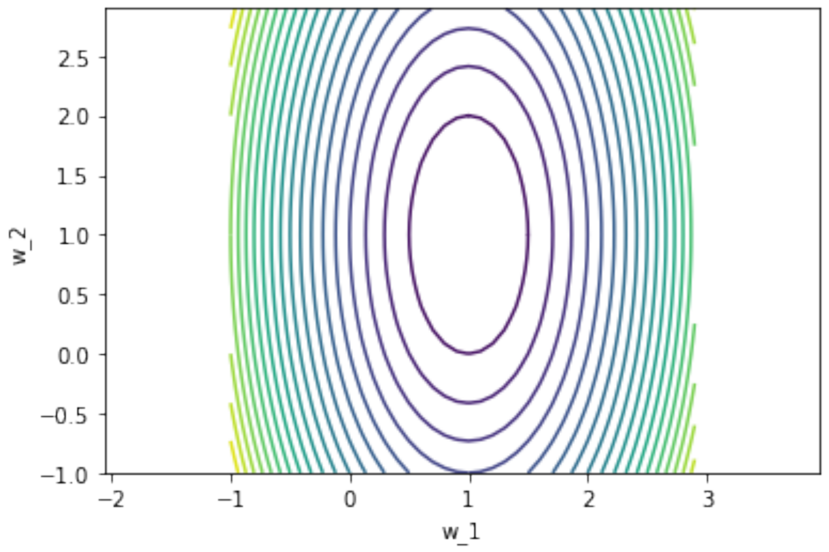
## Questiopn 1a)

when w = w_LS, the first term of f(w) becomes 0 and f(w) = c Since the first term is a matrix multiplied by the transpose of itself, it is positive semi-definite

In [4]:
```
## 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
```

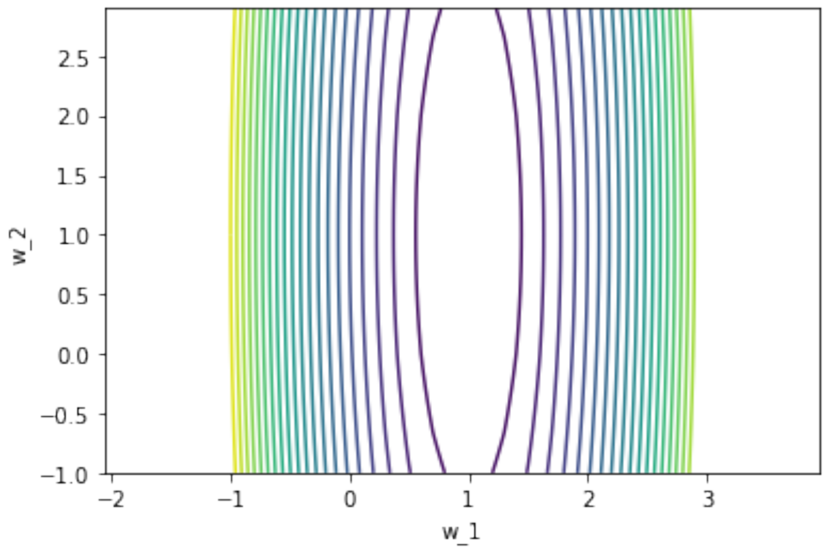
## Question 1b)

In [5]:
```
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

plt.contour(w1,w2,fw,20)
plt.xlim([-1,3])
plt.xlabel('w_1')
plt.ylim([-1,3])
plt.ylabel('w_2');
plt.axis('equal');
```



## Question 1c)

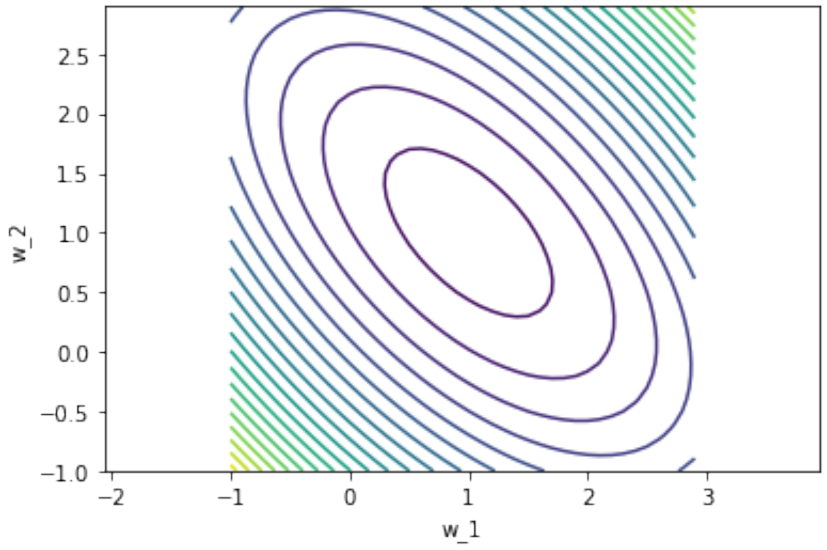
In [6]:
```
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 1/5]])
Sinv = np.linalg.inv(S)
V = np.eye(2)
X = U @ S @ V.T
y = np.array([[1], [1/5], [1], [0]])

### 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

plt.contour(w1,w2,fw,20)
plt.xlim([-1,3])
plt.xlabel('w_1')
plt.ylim([-1,3])
plt.ylabel('w_2');
plt.axis('equal');
```



singular values stretche the countour

## Question 1d)

In [8]:
```
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 1/2]])
Sinv = np.linalg.inv(S)
V = 1/(2**.5) * np.array([[1, 1], [1, -1]])
X = U @ S @ V.T
y = np.array([[2**.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


plt.contour(w1,w2,fw,20)
plt.xlim([-1,3])
plt.xlabel('w_1')
plt.ylim([-1,3])
plt.ylabel('w_2');
plt.axis('equal');
```

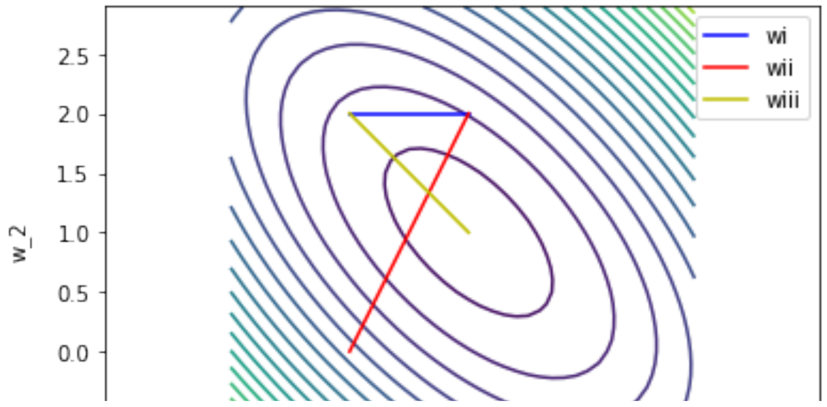


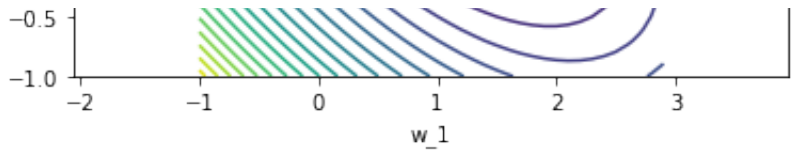
singular vectors rotate the countours

## Question 1e)

In [19]:
```
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 1/2]])
Sinv = np.linalg.inv(S)
V = 1/(2**.5) * np.array([[1, 1], [1, -1]])
X = U @ S @ V.T
y = np.array([[2**.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 vectors
plt.plot([2, 2], c='b',label='wi')
plt.plot([0, 2], c='r',label='wii')
plt.plot([2, 1], c='y',label='wiii')

plt.contour(w1,w2,fw,20)
plt.xlim([-1,3])
plt.xlabel('w_1')
plt.ylim([-1,3])
plt.ylabel('w_2');
plt.axis('equal');
plt.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x11fa68950>
```





## Question 2a)

t_max = 2 / ||X||op^2 = 2 / first singular value^2 = 2

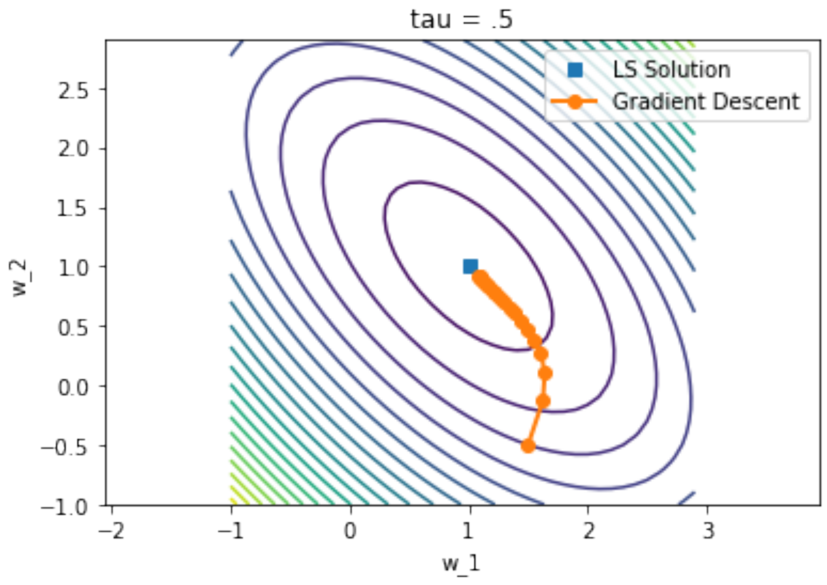
## Question 2b)

In [22]:
```
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
```

### Part 1

```
w_init =  [1.5]
          [-.5]
```

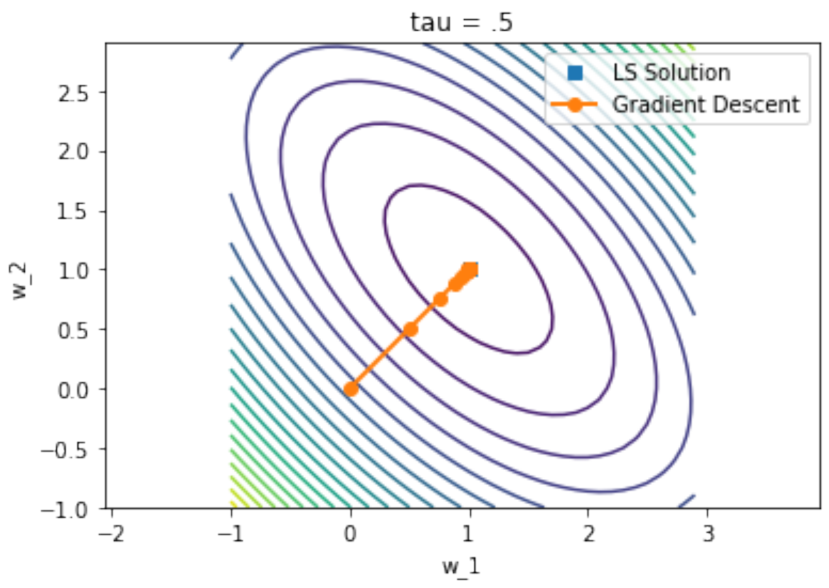
In [23]:
```
w_init = np.array([[1.5], [-.5]])
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);

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('tau = .5');
plt.axis('equal');
```



## Part 2

```
w_init =  [0]
          [0]
```

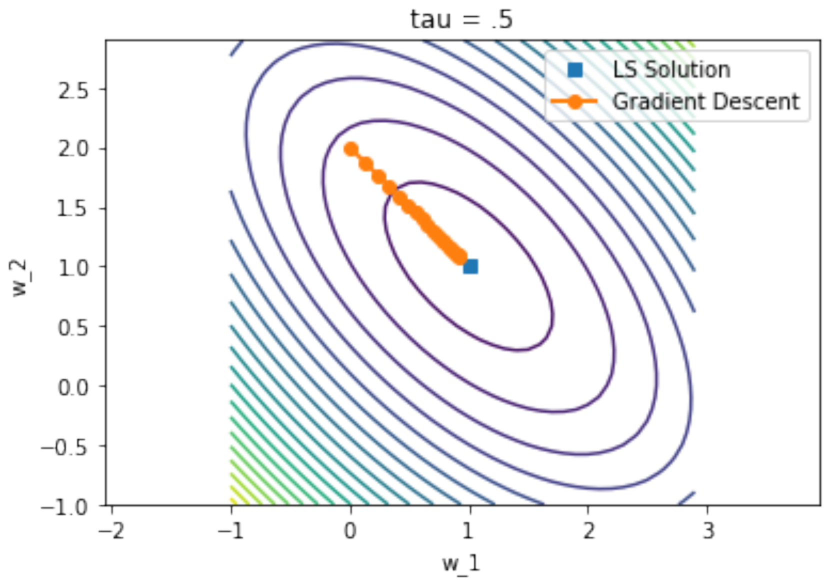
In [24]:
```
w_init = np.array([[0], [0]])
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);

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('tau = .5');
plt.axis('equal');
```



## Part 3

```
w_init =  [0]
          [2]
```

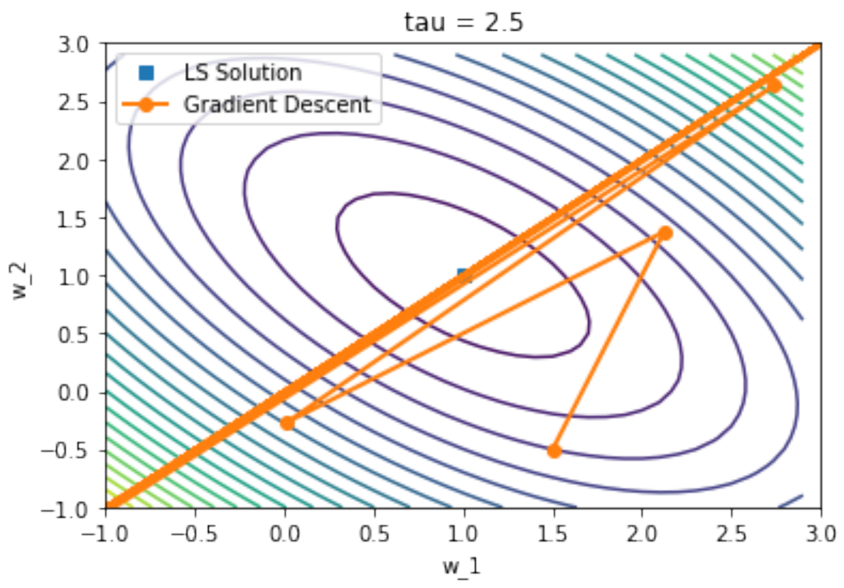
In [25]:
```
w_init = np.array([[0], [2]])
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);

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('tau = .5');
plt.axis('equal');
```



Here the step size is small enough. At each stage, at its countour, it moves to the direction of the negative gradient (orthogonal to the countour). Eventually it will arrive at minimum.

## Question 2c)

In [26]:
```
w_init = np.array([[1.5], [-.5]])
it = 20
tau = 2.5
W = graddescent(X,y,tau,w_init,it);

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('tau = 2.5');
#plt.axis('equal');
```

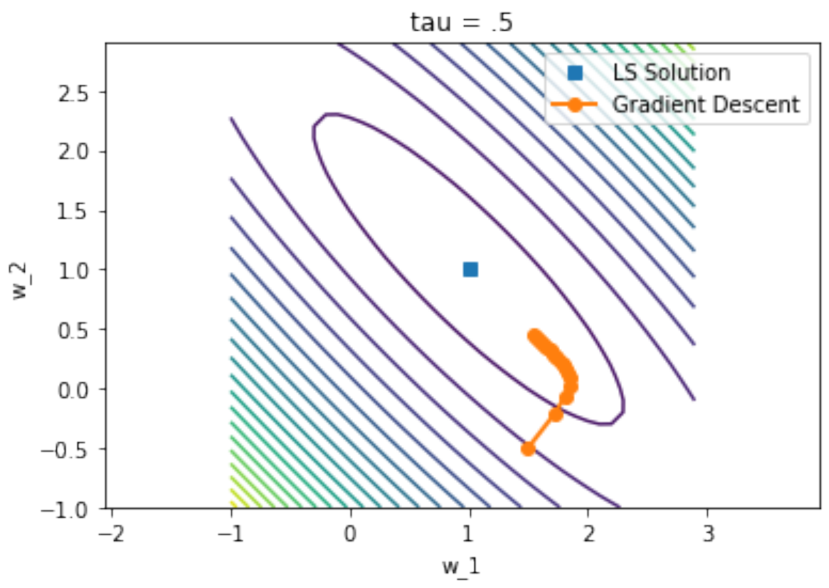


The stepsize is too large.

## Question 2d)

In [30]:
```
U = np.array([[1, 0], [0, 1], [0, 0], [0, 0]])
S = np.array([[1, 0], [0, 1/4]])
Sinv = np.linalg.inv(S)
V = 1/(2**.5) * 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)
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 [31]:
```
w_init = np.array([[1.5], [-0.5]])
it = 20
tau = .5
W = graddescent(X,y,tau,w_init,it);

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('tau = .5');
plt.axis('equal');
```



the step size is too small, an infinitly many steps are needed to reach the LS solution. Further decreasing cost function makes it harder to reach minimum.

## Question 2e

When the ration of sv's increase, the number of iterations needed to reach optimum increase.

In [ ]: