1 实现Gradient Descent

In [6]:

```
x_data = [338.,333.,328.,207.,226.,25.,179.,60.,208.,606.]
y_data = [640.,633.,619.,393.,428.,27.,193.,60.,226.,1591.]
# ydata = b+ w* xdata
print("x_data: ",x_data,len(x_data))
print("y_data: ",y_data)
```

```
x_data:  [338.0, 333.0, 328.0, 207.0, 226.0, 25.0, 179.0, 60.0, 208.0, 606.0] 10
y_data:  [640.0, 633.0, 619.0, 393.0, 428.0, 27.0, 193.0, 60.0, 226.0, 1591.0]
```

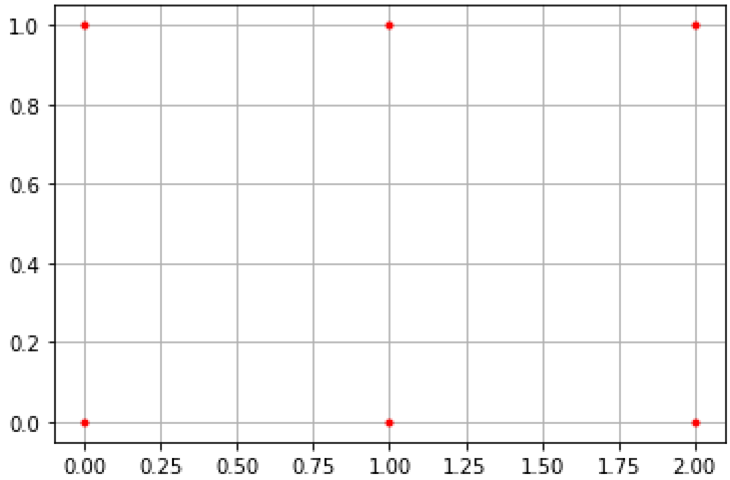
In [2]:

```
import numpy as np


x = np.arange(-200,-100,1) # bias
y = np.arange(-5,5,0.1) # weights
X,Y = np.meshgrid(x,y)
```

1.1 解释 np.meshgrid的作用

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
x = np.array([[0, 1, 2], [0, 1, 2]])
y = np.array([[0, 0, 0], [1, 1, 1]])


plt.plot(x, y,
         color='red',  # 全部点设置为红色
         marker='.',  # 点的形状为圆点
         linestyle='')  # 线型为空，也即点与点之间不用线连接
plt.grid(True)
plt.show()
```



语法： X，Y = np.meshgrid(x,y)

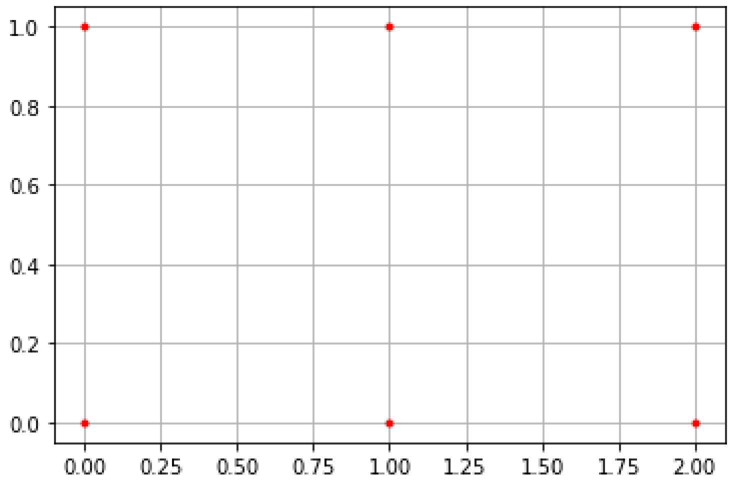
输入x,y就是网格点的横纵坐标列列向量(非矩阵)

输出的x,y,就是坐标矩阵

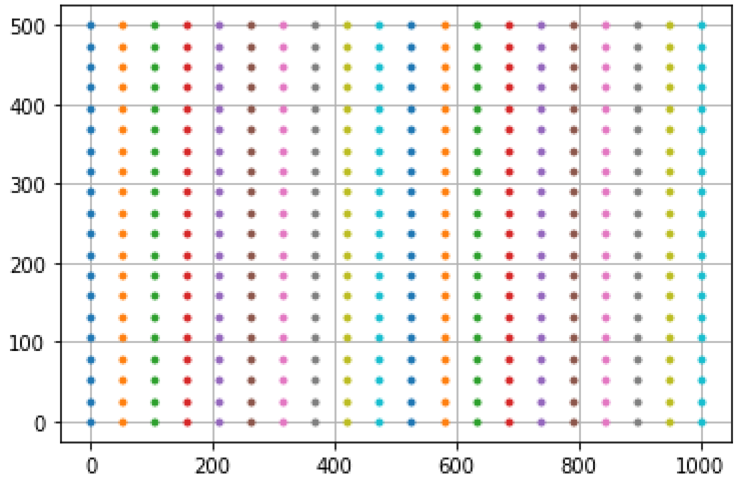
In [7]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

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

X,Y = np.meshgrid(x,y)
print(X,'\n',Y)
plt.plot(X, Y,
         color='red',  # 全部点设置为红色
         marker='.',  # 点的形状为圆点
         linestyle='')  # 线型为空，也即点与点之间不用线连接
plt.grid(True)
plt.show()

x = np.linspace(0,1000,20)
y = np.linspace(0,500,20)
print(x,'\n',y)
X,Y = np.meshgrid(x,y)
plt.plot(X,Y,
        marker='.',
        linestyle='')
plt.grid(True)
plt.show()
```

```
[[0 1 2]
 [0 1 2]]
 [[0 0 0]
 [1 1 1]]
```




```
[   0.          52.63157895  105.26315789  157.89473684  210.52631579
  263.15789474  315.78947368  368.42105263  421.05263158  473.68421053
  526.31578947  578.94736842  631.57894737  684.21052632  736.84210526
  789.47368421  842.10526316  894.73684211  947.36842105 1000.        ]
 [  0.          26.31578947  52.63157895  78.94736842 105.26315789
 131.57894737 157.89473684 184.21052632 210.52631579 236.84210526
 263.15789474 289.47368421 315.78947368 342.10526316 368.42105263
 394.73684211 421.05263158 447.36842105 473.68421053 500.        ]
```



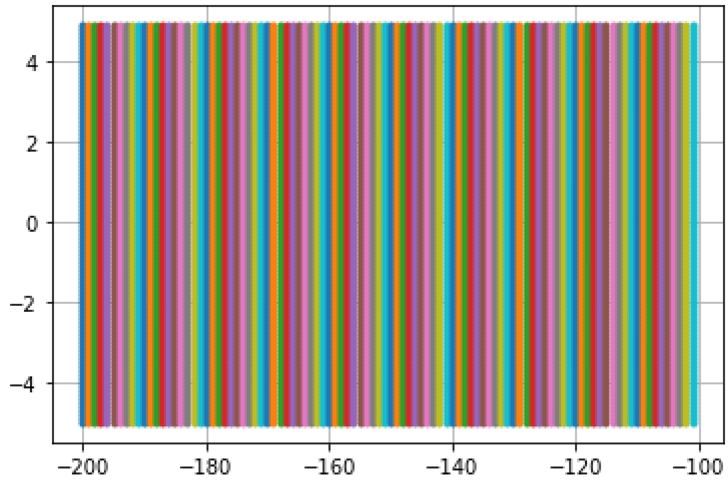
1.2 继续实现 Gradient Descent

In [11]:

```
import numpy as np
x_data = [338.,333.,328.,207.,226.,25.,179.,60.,208.,606.]
y_data = [640.,633.,619.,393.,428.,27.,193.,60.,226.,1591.]
# ydata = b+ w* xdata
# print("x_data: ",x_data,len(x_data))
# print("y_data: ",y_data)

x = np.arange(-200,-100,1) # bias
y = np.arange(-5,5,0.1) # weights
print(x,'\n',y,len(x),len(y))
X,Y = np.meshgrid(x,y)
print(X,'\n',Y)
plt.plot(X,Y,
        marker='.',
        linestyle='')
plt.grid(True)
plt.show()
```

```
[-200 -199 -198 -197 -196 -195 -194 -193 -192 -191 -190 -189 -188 -187
 -186 -185 -184 -183 -182 -181 -180 -179 -178 -177 -176 -175 -174 -173
 -172 -171 -170 -169 -168 -167 -166 -165 -164 -163 -162 -161 -160 -159
 -158 -157 -156 -155 -154 -153 -152 -151 -150 -149 -148 -147 -146 -145
 -144 -143 -142 -141 -140 -139 -138 -137 -136 -135 -134 -133 -132 -131
 -130 -129 -128 -127 -126 -125 -124 -123 -122 -121 -120 -119 -118 -117
 -116 -115 -114 -113 -112 -111 -110 -109 -108 -107 -106 -105 -104 -103
 -102 -101]
 [-5.00000000e+00 -4.90000000e+00 -4.80000000e+00 -4.70000000e+00
 -4.60000000e+00 -4.50000000e+00 -4.40000000e+00 -4.30000000e+00
 -4.20000000e+00 -4.10000000e+00 -4.00000000e+00 -3.90000000e+00
 -3.80000000e+00 -3.70000000e+00 -3.60000000e+00 -3.50000000e+00
 -3.40000000e+00 -3.30000000e+00 -3.20000000e+00 -3.10000000e+00
 -3.00000000e+00 -2.90000000e+00 -2.80000000e+00 -2.70000000e+00
 -2.60000000e+00 -2.50000000e+00 -2.40000000e+00 -2.30000000e+00
 -2.20000000e+00 -2.10000000e+00 -2.00000000e+00 -1.90000000e+00
 -1.80000000e+00 -1.70000000e+00 -1.60000000e+00 -1.50000000e+00
 -1.40000000e+00 -1.30000000e+00 -1.20000000e+00 -1.10000000e+00
 -1.00000000e+00 -9.00000000e-01 -8.00000000e-01 -7.00000000e-01
 -6.00000000e-01 -5.00000000e-01 -4.00000000e-01 -3.00000000e-01
 -2.00000000e-01 -1.00000000e-01 -1.77635684e-14  1.00000000e-01
  2.00000000e-01  3.00000000e-01  4.00000000e-01  5.00000000e-01
  6.00000000e-01  7.00000000e-01  8.00000000e-01  9.00000000e-01
  1.00000000e+00  1.10000000e+00  1.20000000e+00  1.30000000e+00
  1.40000000e+00  1.50000000e+00  1.60000000e+00  1.70000000e+00
  1.80000000e+00  1.90000000e+00  2.00000000e+00  2.10000000e+00
  2.20000000e+00  2.30000000e+00  2.40000000e+00  2.50000000e+00
  2.60000000e+00  2.70000000e+00  2.80000000e+00  2.90000000e+00
  3.00000000e+00  3.10000000e+00  3.20000000e+00  3.30000000e+00
  3.40000000e+00  3.50000000e+00  3.60000000e+00  3.70000000e+00
  3.80000000e+00  3.90000000e+00  4.00000000e+00  4.10000000e+00
  4.20000000e+00  4.30000000e+00  4.40000000e+00  4.50000000e+00
  4.60000000e+00  4.70000000e+00  4.80000000e+00  4.90000000e+00] 100 100
[[-200 -199 -198 ... -103 -102 -101]
 [-200 -199 -198 ... -103 -102 -101]
 [-200 -199 -198 ... -103 -102 -101]
 ...
 [-200 -199 -198 ... -103 -102 -101]
 [-200 -199 -198 ... -103 -102 -101]
 [-200 -199 -198 ... -103 -102 -101]]
 [[-5.  -5.  -5.  ... -5.  -5.  -5. ]
```

```
 [-4.9 -4.9 -4.9 ... -4.9 -4.9 -4.9]
 [-4.8 -4.8 -4.8 ... -4.8 -4.8 -4.8]
 ...
 [ 4.7  4.7  4.7 ...  4.7  4.7  4.7]
 [ 4.8  4.8  4.8 ...  4.8  4.8  4.8]
 [ 4.9  4.9  4.9 ...  4.9  4.9  4.9]]
```



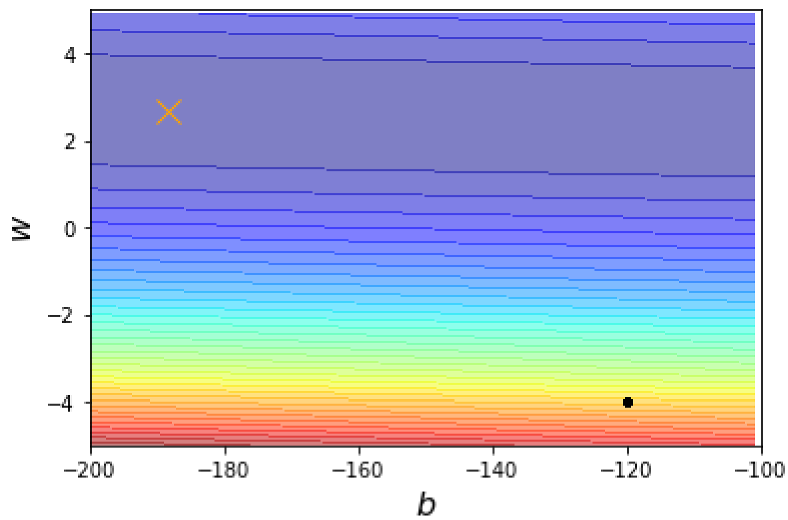
In [15]:

```
import numpy as np
x_data = [338., 333., 328., 207., 226., 25., 179., 60., 208., 606.]
y_data = [640., 633., 619., 393., 428., 27., 193., 60., 226., 1591.]
# ydata = b+ w* xdata
print("x_data: ",x_data,len(x_data))
print("y_data: ",y_data)

x = np.arange(-200,-100,1) # bias 100
y = np.arange(-5,5,0.1) # weights 100
z = np.zeros((len(x),len(y))) # 100*100
X,Y = np.meshgrid(x,y)
for i in range(len(x)):
    for j in range(len(y)):
        b = x[i]
        w = y[j]
        z[j][i]=0
        for n in range(len(x_data)):
            z[j][i] = z[j][i] + (y_data[n] - b - w*x_data[n]) ** 2
        z[j][i] = z[j][i]/len(x_data)
```

```
x_data:  [338.0, 333.0, 328.0, 207.0, 226.0, 25.0, 179.0, 60.0, 208.0, 606.0] 10
y_data:  [640.0, 633.0, 619.0, 393.0, 428.0, 27.0, 193.0, 60.0, 226.0, 1591.0]
```

In [34]:

```
# y_data = b + w*x_data
b = -120
w = -4
lr = 0.000001
iteration = 1000000

b_history = [b]
w_history = [w]

lr_b = 0
lr_w = 0

for i in range(iteration):
    b_grad = 0.0
    w_grad = 0.0
    for n in range(len(x_data)):
        b_grad = b_grad - 2.0*(y_data[n] - b - w*x_data[n]) * 1.0
        w_grad = w_grad - 2.0*(y_data[n] - b - w*x_data[n])*x_data[n]
    # Adagrad
    lr_b = lr_b + b_grad ** 2
    lr_w = lr_w + w_grad ** 2

    b = b - lr/np.sqrt(lr_b) * b_grad
    w = w - lr/np.sqrt(lr_w) * w_grad
#     b = b - lr * b_grad
#     w = w - lr * w_grad


    b_history.append(b)
    w_history.append(w)

plt.contourf(x,y,z,50,alpha=0.5,cmap=plt.get_cmap('jet'))
plt.plot([-188.4],[2.67],'x',ms=12,lw=1.5,color='orange')
plt.plot(b_history,w_history,'o-',ms=3,lw=1.5,color='black')
plt.xlim(-200,-100)
plt.ylim(-5,5)
plt.xlabel(r'$b$',fontsize=16)
plt.ylabel(r'$w$',fontsize=16)
plt.show()
```



In [ ]:

```

```