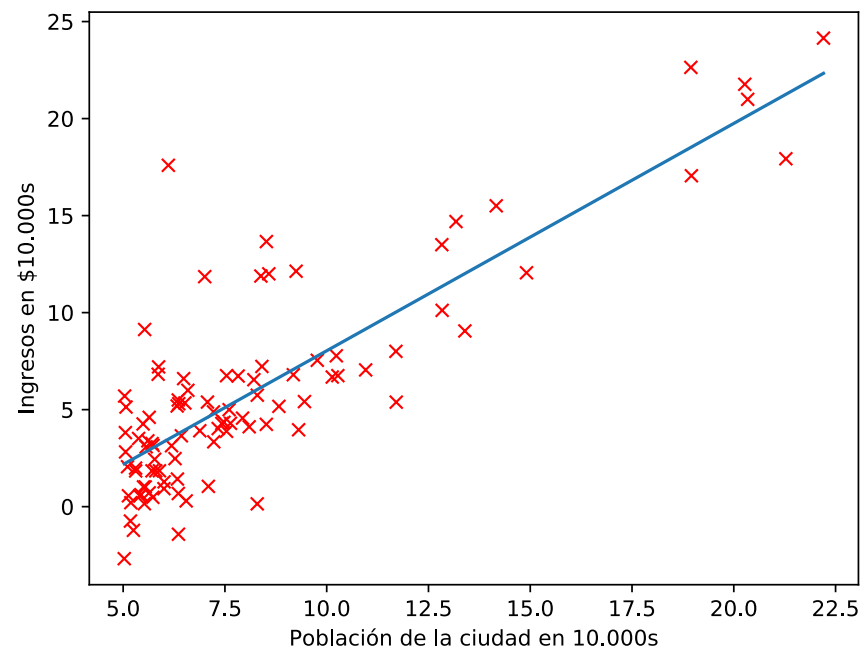
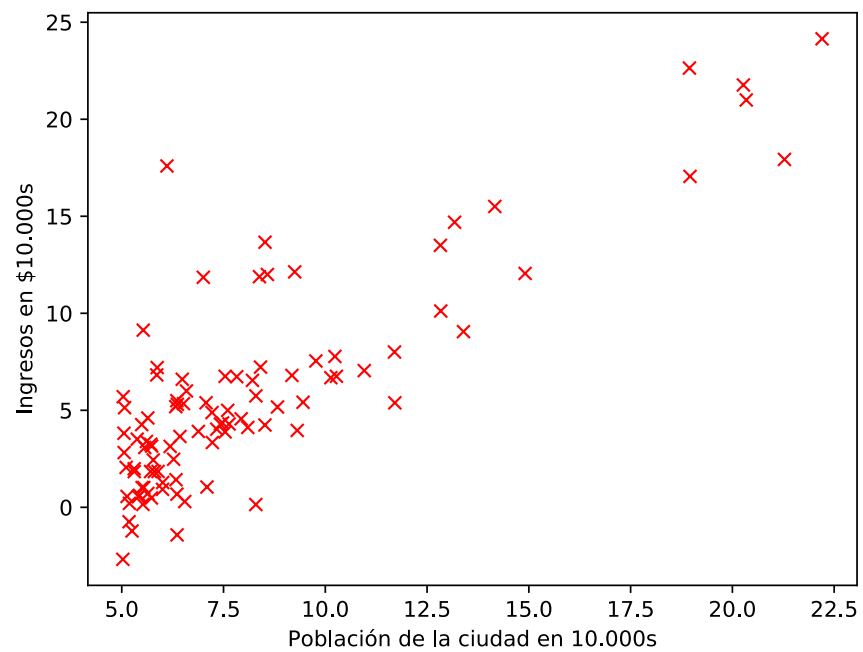


Práctica 0: vectorización

```
def integra_mc_vec(fun, a, b, num_puntos=100):  
    """Calcula la integral de f entre a y b por el método de  
    Monte Carlo sin usar bucles  
    """  
  
    eje_x = np.linspace(a, b, num_puntos)  
    eje_y = fun(eje_x)  
    max_f = max(eje_y)  
    min_f = 0  
    random_x = np.random.uniform(a, b, num_puntos)  
    random_y = np.random.uniform(min_f, max_f, num_puntos)  
    f_random_x = fun(random_x)  
  
    plt.figure()  
    plt.plot(random_x, random_y, 'x', c='red')  
    plt.plot(eje_x, eje_y, '-')  
    plt.savefig('mc.pdf')  
    plt.close()  
  
    debajo = sum(random_y < f_random_x)  
    area_total = abs(b - a) * abs(max_f - min_f)  
    return area_total * (debajo / num_puntos)
```

Práctica 1:

regresión lineal



```
import numpy as np
from pandas.io.parsers import read_csv

def carga_csv(file_name):
    """carga el fichero csv especificado y lo
        devuelve en un array de numpy
    """
    valores = read_csv(file_name, header=None).values
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)
```

```
datos = carga_csv('ex1data1.csv')
```

```
X = datos[:, :-1]  
np.shape(X)          # (97, 1)  
Y = datos[:, -1]  
np.shape(Y)          # (97,)
```

```
m = np.shape(X)[0]  
n = np.shape(X)[1]
```

```
# añadimos una columna de 1's a la X  
X = np.hstack([np.ones([m, 1]), X])
```

```
alpha = 0.01  
Thetas, costes = descenso_gradiente(X, Y, alpha)
```

Gradient descent algorithm

$$\left. \begin{array}{l} \text{repeat until convergence} \{ \\ \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \\ \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \end{array} \right\} \begin{array}{l} \text{update} \\ \theta_0 \text{ and } \theta_1 \\ \text{simultaneously} \end{array}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$h_{\theta}(x) = \theta^T x$$

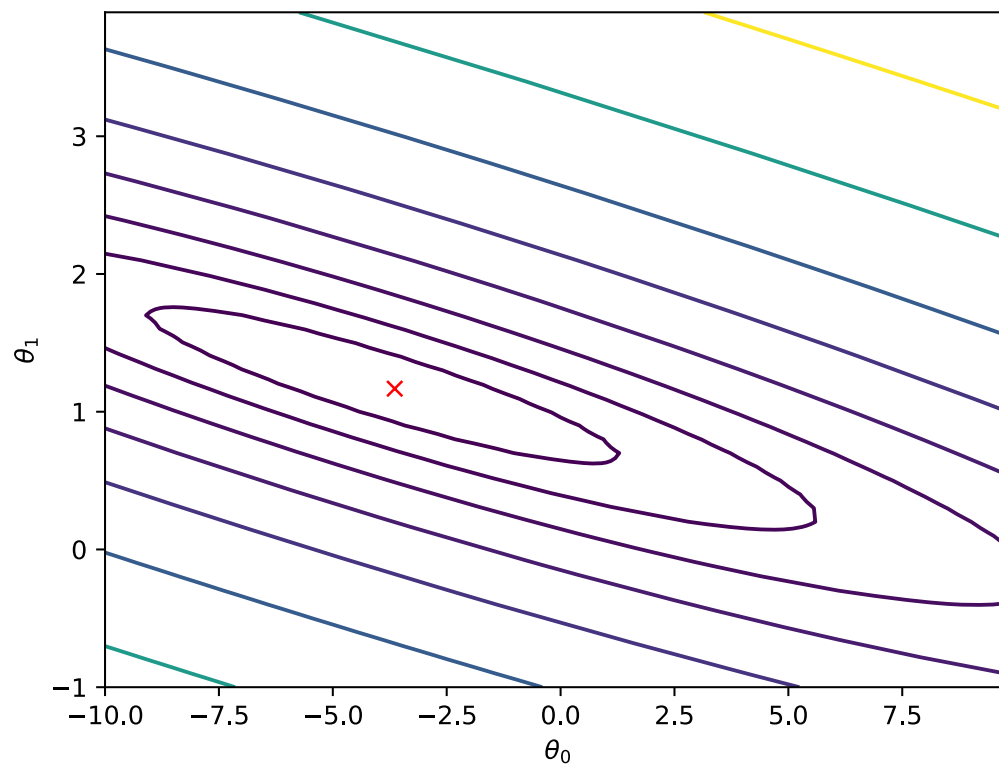
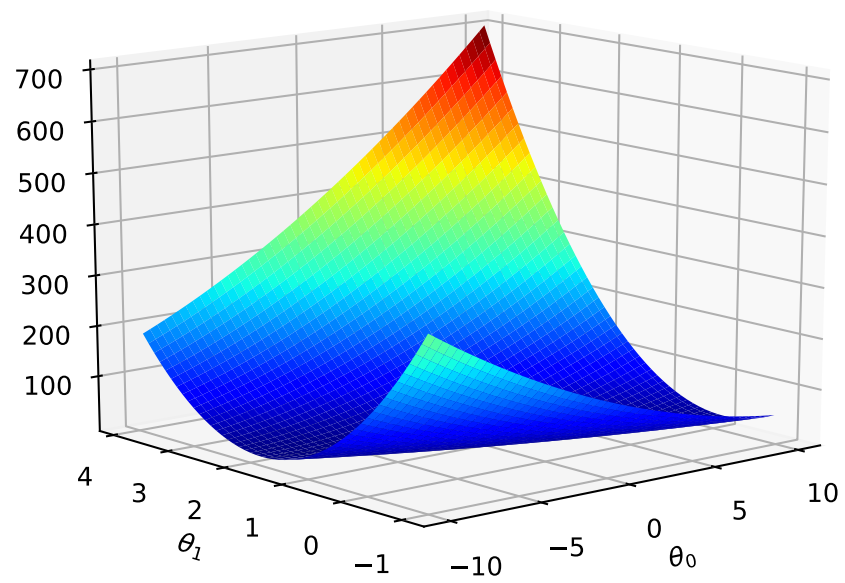
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$X = \begin{bmatrix} \text{---} & (x^{(1)})^T & \text{---} \\ \text{---} & (x^{(2)})^T & \text{---} \\ & \vdots & \\ \text{---} & (x^{(m)})^T & \text{---} \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

```
def coste(X, Y, Theta):
    H = np.dot(X, Theta)
    Aux = (H - Y) ** 2
    return Aux.sum() / (2 * len(X))
```

```
import numpy as np
```

```
In [3]: x = np.array([1,2,3])
```

```
In [4]: y = np.array([4,5,6])
```

```
In [5]: xx, yy = np.meshgrid(x,y)
```

```
In [6]: xx
```

```
Out[6]:
```

```
array([[1, 2, 3],  
       [1, 2, 3],  
       [1, 2, 3]])
```

```
In [7]: yy
```

```
Out[7]:
```

```
array([[4, 4, 4],  
       [5, 5, 5],  
       [6, 6, 6]])
```

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np
```

```
fig = plt.figure()
ax = fig.gca(projection='3d')    # ax = Axes3D(fig)
```

```
# Make data.
```

```
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
```

```
# Plot the surface.
```

```
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
```

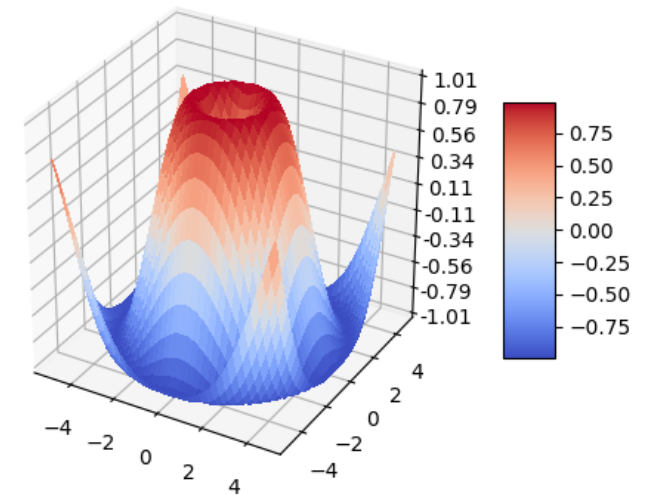
```
# Customize the z axis.
```

```
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
```

```
# Add a color bar which maps values to colors.
```

```
fig.colorbar(surf, shrink=0.5, aspect=5)
```

```
plt.show()
```

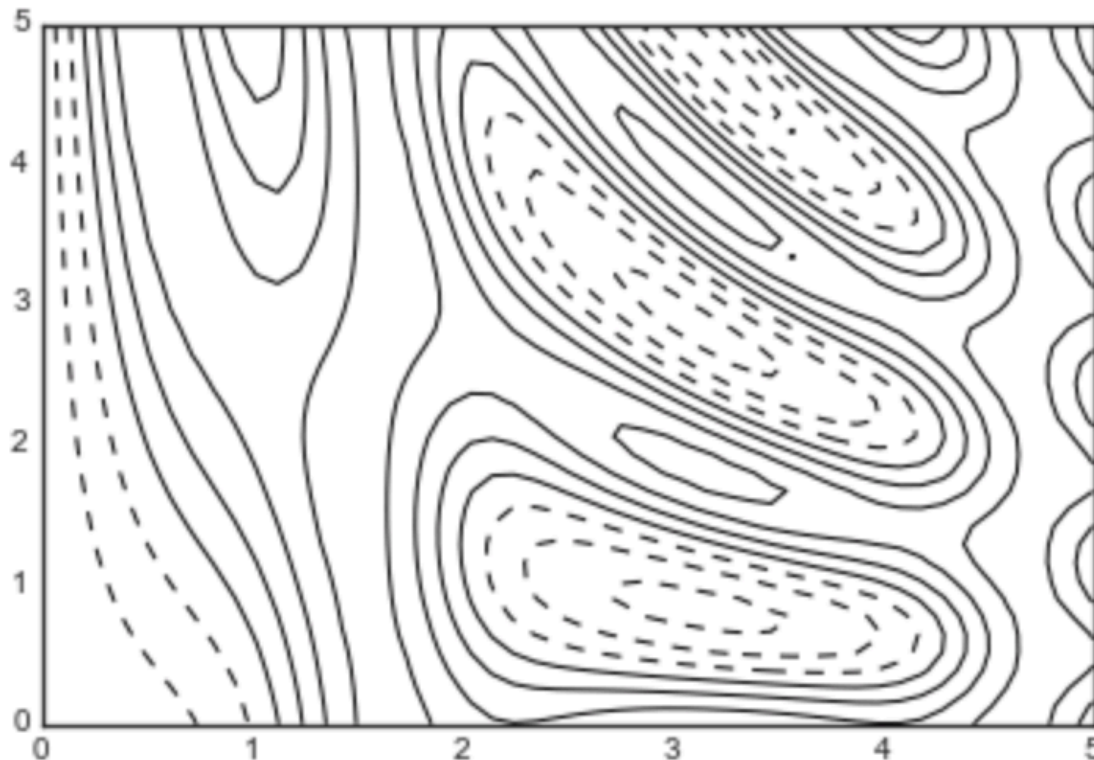


```
def f(x, y):  
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

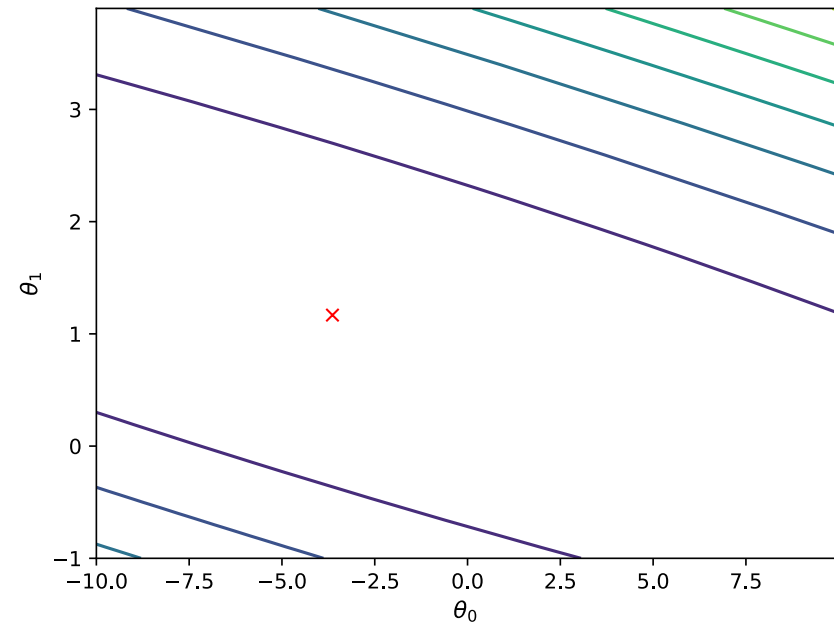
```
x = np.linspace(0, 5, 50)  
y = np.linspace(0, 5, 40)
```

```
X, Y = np.meshgrid(x, y)  
Z = f(X, Y)
```

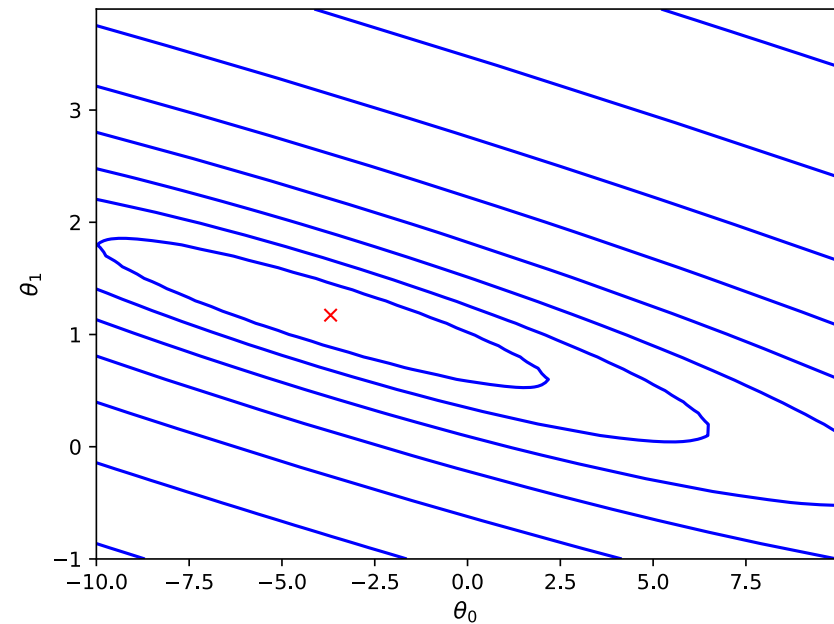
```
plt.contour(X, Y, Z, colors='black');
```



```
plt.contour(Theta0, Theta1, Coste)
```



```
plt.contour(Theta0, Theta1, Coste,  
            np.logspace(-2, 3, 20), colors='blue')
```

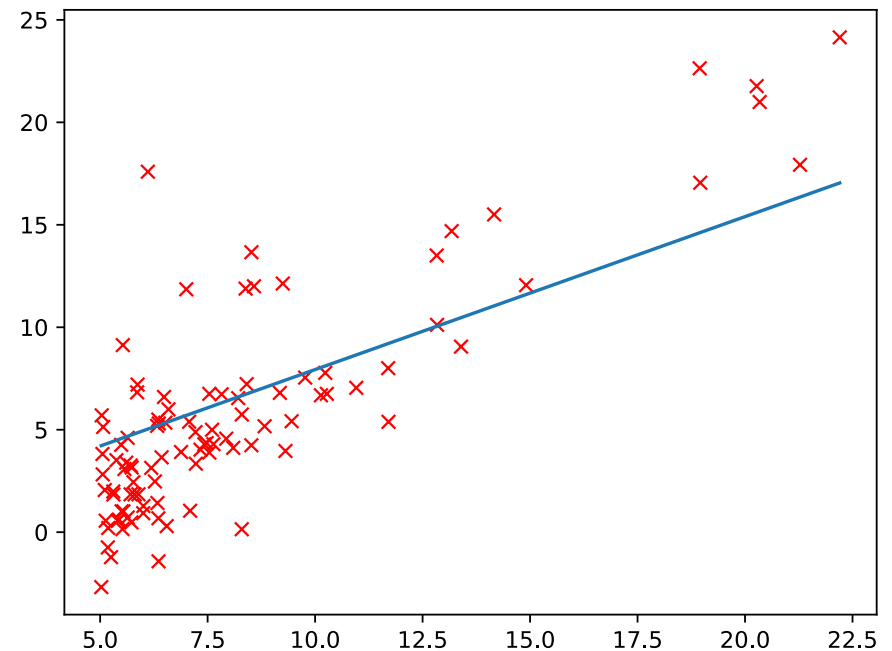


```
def make_data(t0_range, t1_range, X, Y):  
    """Genera las matrices X,Y,Z para generar un plot en 3D  
    """  
  
    step = 0.1  
    Theta0 = np.arange(t0_range[0], t0_range[1], step)  
    Theta1 = np.arange(t1_range[0], t1_range[1], step)  
  
    Theta0, Theta1 = np.meshgrid(Theta0, Theta1)  
    # Theta0 y Theta1 tienen las mismas dimensiones, de forma que  
    # cogiendo un elemento de cada uno se generan las coordenadas x,y  
    # de todos los puntos de la rejilla  
  
    Coste = np.empty_like(Theta0)  
    for ix, iy in np.ndindex(Theta0.shape):  
        Coste[ix, iy] = coste(X, Y, [Theta0[ix, iy], Theta1[ix, iy]])  
  
    return [Theta0, Theta1, Coste]
```

```

def gradiente(X, Y, Theta, alpha):
    NuevaTheta = Theta
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    H = np.dot(X, Theta)
    Aux = (H - Y)
    for i in range(n):
        Aux_i = Aux * X[:, i]
        NuevaTheta -= (alpha / m) * Aux_i.sum()
    return NuevaTheta

```



```
def gradiente(X, Y, Theta, alpha):  
    NuevaTheta = Theta  
    m = np.shape(X)[0]  
    n = np.shape(X)[1]  
    H = np.dot(X, Theta)  
    Aux = (H - Y)  
    for i in range(n):  
        Aux_i = Aux * X[:, i]  
        NuevaTheta[i] -= (alpha / m) * Aux_i.sum()  
    return NuevaTheta
```

