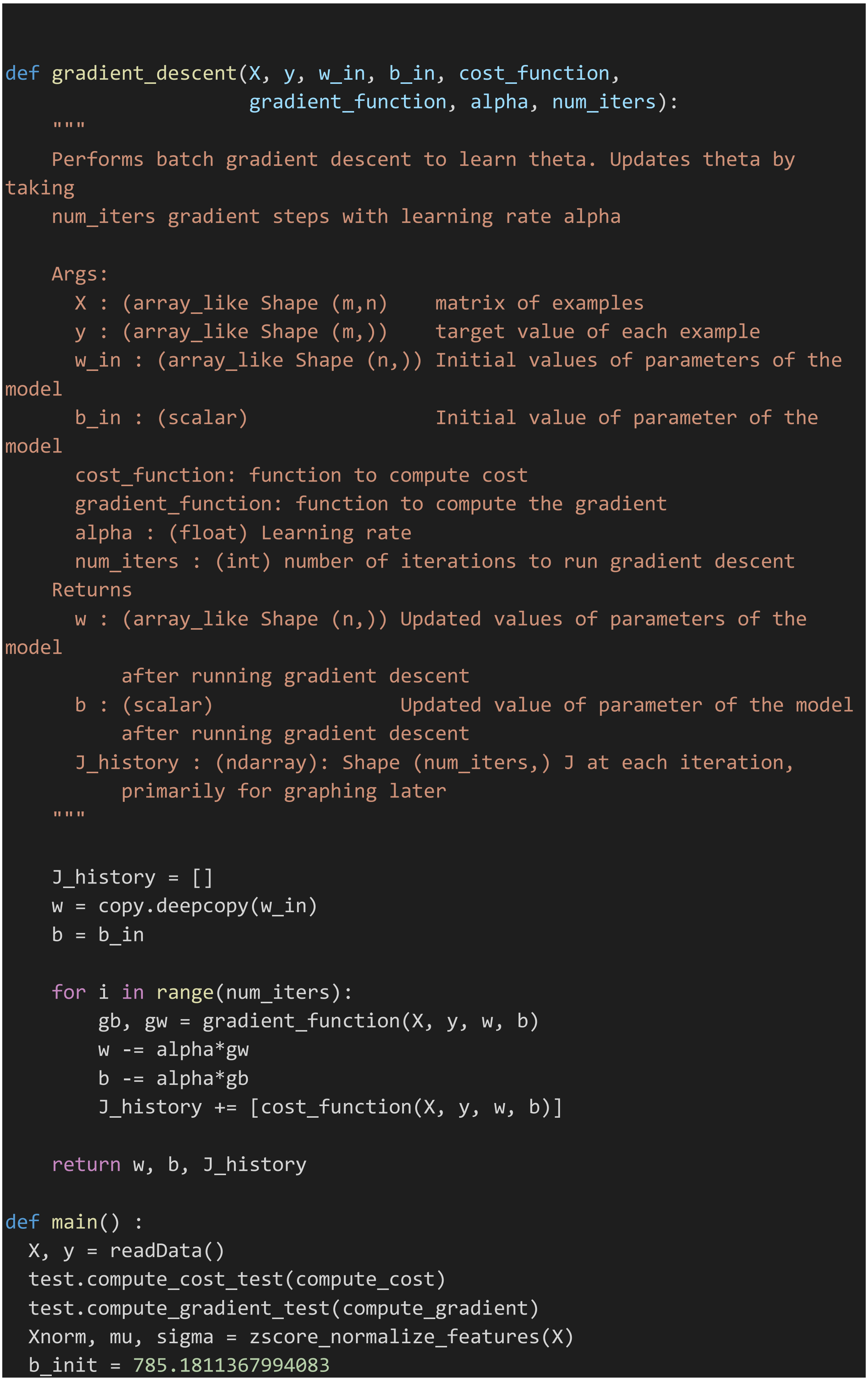
Práctica 2: Regresión lineal multi-variable

Alberto Muñoz Fernández

Óscar García Castro

```python
import numpy as np
import copy
import math
import matplotlib.pyplot as plt
import public_tests as test

def visualize_data(X_train, y_train , y_pre):
  X_features=['size(sqft)','bedrooms','floors','age']
  fig,ax=plt.subplots(1,4,figsize=(25,5),sharey=True)
  for i in range(len(ax)):
    ax[i].scatter(X_train[:,i],y_train)
    ax[i].scatter(X_train[:,i],y_pre)
    ax[i].set_xlabel(X_features[i])
  ax[0].set_ylabel("Price(1000's)")
  plt.show()

def readData():
  data=np.loadtxt("./data/houses.txt",delimiter=',',skiprows=1)
  X_train=data[:,:4]
  y_train=data[:,4]
  return X_train, y_train

def zscore_normalize_features(X):
  """
  computes  X, zcore normalized by column

  Args:
    X (ndarray (m,n))     : input data, m examples, n features

  Returns:
    X_norm (ndarray (m,n)): input normalized by column
    mu (ndarray (n,))     : mean of each feature
    sigma (ndarray (n,))  : standard deviation of each feature
  """
  X_norm = np.zeros((len(X), len(X[0])))

  sigma = np.std(X, axis = 0)
  mu = np.mean(X, axis = 0)

  for i in range(len(X)) :
    X_norm[i] = (X[i] - mu)/sigma

  return (X_norm, mu, sigma)


def compute_cost(X, y, w, b):

  """

  compute cost
  Args:
    X (ndarray (m,n)): Data, m examples with n features
    y (ndarray (m,)) : target values
    w (ndarray (n,)) : model parameters
    b (scalar)       : model parameter
  Returns
    cost (scalar)    : cost
  """
  cost = 0

  for i in range(len(X)):
    cost += ((np.dot(X[i], w)+ b) - y[i])**2

  cost /= (2 * len(X))
  return cost


def compute_gradient(X, y, w, b):
    """
    Computes the gradient for linear regression
    Args:
      X : (ndarray Shape (m,n)) matrix of examples
      y : (ndarray Shape (m,))  target value of each example
      w : (ndarray Shape (n,))  parameters of the model
      b : (scalar)              parameter of the model
    Returns
      dj_dw : (ndarray Shape (n,)) The gradient of the cost w.r.t. the
parameters w.
      dj_db : (scalar)             The gradient of the cost w.r.t. the
parameter b.
    """
    dj_db = 0
    m = len(X)
    dj_dw = np.zeros(len(X[0]))

    for i in range (m):
      dj_db += np.dot(w, X[i]) + b - y[i]
      dj_dw += (np.dot(w, X[i]) + b - y[i]) * X[i]

    return dj_db/m, dj_dw/m
```



```
def gradient_descent(X, y, w_in, b_in, cost_function,
                     gradient_function, alpha, num_iters):
    """
    Performs batch gradient descent to learn theta. Updates theta by
taking
    num_iters gradient steps with learning rate alpha

    Args:
      X : (array_like Shape (m,n)    matrix of examples
      y : (array_like Shape (m,))    target value of each example
      w_in : (array_like Shape (n,)) Initial values of parameters of the
model
      b_in : (scalar)                Initial value of parameter of the
model
      cost_function: function to compute cost
      gradient_function: function to compute the gradient
      alpha : (float) Learning rate
      num_iters : (int) number of iterations to run gradient descent
    Returns
      w : (array_like Shape (n,)) Updated values of parameters of the
model
          after running gradient descent
      b : (scalar)                Updated value of parameter of the model
          after running gradient descent
      J_history : (ndarray): Shape (num_iters,) J at each iteration,
          primarily for graphing later
    """

    J_history = []
    w = copy.deepcopy(w_in)
    b = b_in

    for i in range(num_iters):
        gb, gw = gradient_function(X, y, w, b)
        w -= alpha*gw
        b -= alpha*gb
        J_history += [cost_function(X, y, w, b)]

    return w, b, J_history

def main() :
  X, y = readData()
  test.compute_cost_test(compute_cost)
  test.compute_gradient_test(compute_gradient)
  Xnorm, mu, sigma = zscore_normalize_features(X)
  b_init = 785.1811367994083
  w_init = np.array([0.39133535, 18.75376741, -53.36032453, -
26.42131618])
  w, b, J_hist = gradient_descent(Xnorm, y, w_init, b_init, compute_cost,
compute_gradient, 0.1, 1500)
  Ypre = np.dot(Xnorm, w)+ b
  visualize_data(X, y, Ypre)

  x = np.array([1200.0, 3.0, 1.0, 40.0])
  x = (x - mu)/sigma
  print(np.dot(w, x)+ b)

main()
```