In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
```

In [2]:
```
df = pd.read_csv('HW1.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
M
```

Out[2]: 100

In [3]:
```
df.describe()
```

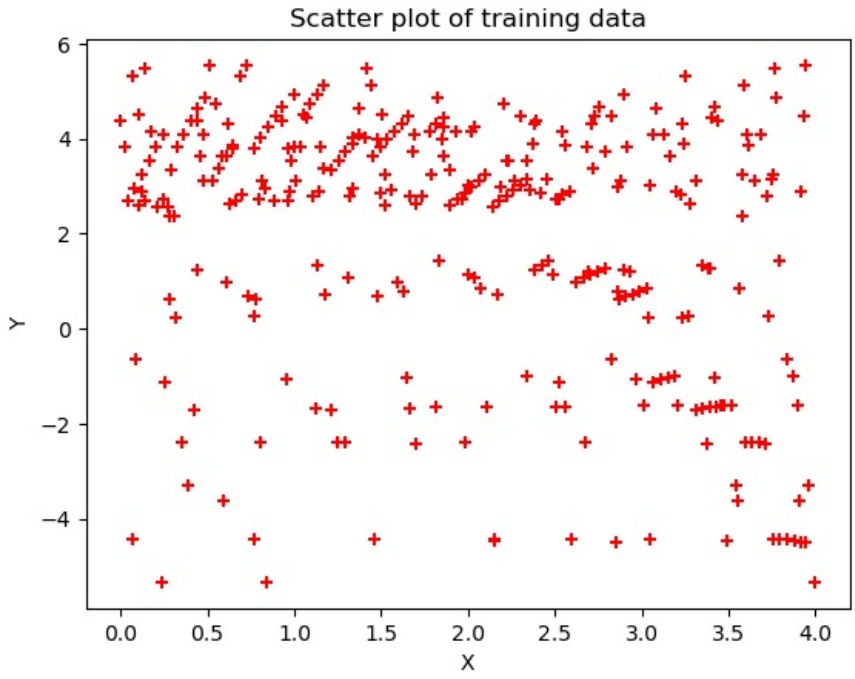
Out[3]:

	X1	X2	X3	Y
count	100.000000	100.000000	100.000000	100.000000
mean	2.000000	2.000000	1.960000	1.851276
std	1.172181	1.172154	1.163005	2.774643
min	0.000000	0.070303	0.027879	-5.332455
25%	1.000000	0.979394	0.952121	0.527533
50%	2.000000	2.009697	1.949091	2.879003
75%	3.000000	3.040000	2.946061	3.925389
max	4.000000	3.949091	3.943030	5.545892

In [4]:
```
# Separate features and labels
X = df.values[:, [0, 1, 2]]  # get input values from first column -- X is a list here
X1 = df.values[:, 0]
X2 = df.values[:, 1]
X3 = df.values[:, 2]
y = df.values[:, [3, 3, 3]]  # get output values from forth column -- Y is the list here
m = len(y)  # Number of training examples
n = len(X)  # Number of training examples


# Display first 5 records and the total number of training examples
print('X = ', X[: 5])
print('y = ', y[: 5])
print('m = ', m)
print('n = ', n)
```

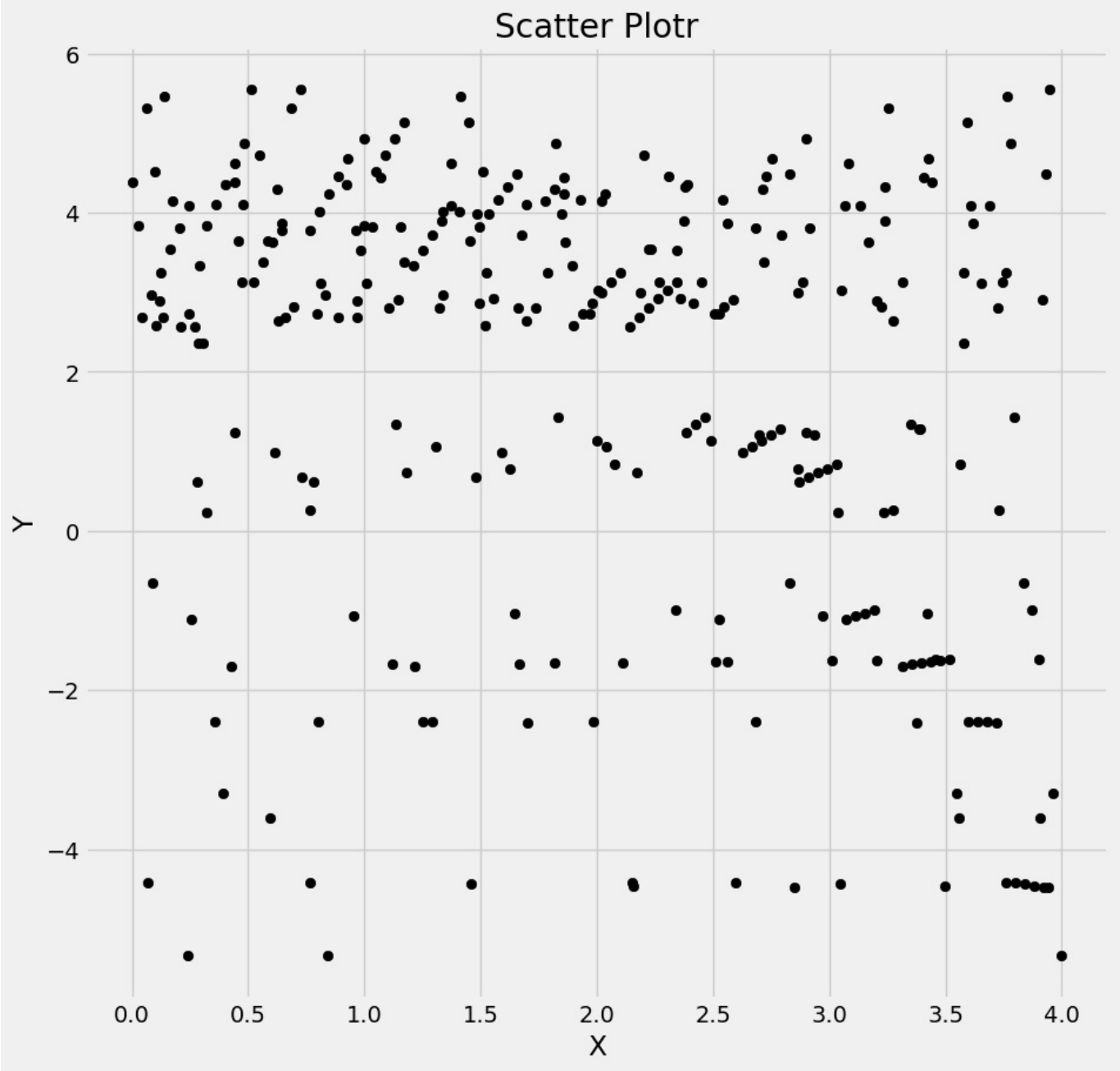
```
X =  [[0.         3.44       0.44      ]
 [0.04040404 0.1349495  0.88848485]
 [0.08080808 0.82989899 1.3369697 ]
 [0.12121212 1.52484848 1.78545454]
 [0.16161616 2.21979798 2.23393939]]
y =  [[4.38754501 4.38754501 4.38754501]
 [2.6796499  2.6796499  2.6796499 ]
 [2.96848981 2.96848981 2.96848981]
 [3.25406475 3.25406475 3.25406475]
 [3.53637472 3.53637472 3.53637472]]
m =  100
n =  100
```

In [5]:
```
# Scatter plot
plt.scatter(X, y, color='red', marker='+')

# Grid, labels, and title
# plt.grid(True)
plt.rcParams["figure.figsize"] = (10, 10)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatter plot of training data')

# Show the plot
plt.show()
```



In [6]:
```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

plt.scatter(X, y, color='black')
plt.xlabel('X')
plt.ylabel('Y')
plt.gca().set_title("Scatter Plotr")
```

Out[6]:
```
Text(0.5, 1.0, 'Scatter Plotr')
```



In [7]:
```
m = len(y)  # Number of training examples
n = len(X)  # Number of training examples
```

In [8]:
```
X_0 = np.ones((m, 1))
X_0[:5]
```

Out[8]:
```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

In [9]:
```
X_1 = X1.reshape(m, 1)
X_1[:10]
```

Out[9]:
```
array([[0.        ],
       [0.04040404],
       [0.08080808],
       [0.12121212],
       [0.16161616],
       [0.2020202 ],
       [0.24242424],
       [0.28282828],
       [0.32323232],
       [0.36363636]])
```

In [10]:
```
X_2 = X2.reshape(m, 1)
X_2[:10]
```

Out[10]:
```
array([[3.44      ],
       [0.1349495 ],
       [0.82989899],
       [1.52484848],
       [2.21979798],
       [2.91474747],
       [3.60969697],
       [0.30464646],
       [0.99959596],
       [1.69454546]])
```

In [11]:
```
X_3 = X3.reshape(m, 1)
X_3[:10]
```

Out[11]:
```
array([[0.44      ],
       [0.88848485],
       [1.3369697 ],
       [1.78545454],
       [2.23393939],
       [2.68242424],
       [3.13090909],
       [3.57939394],
       [0.02787879],
       [0.47636364]])
```

In [12]:
```
# Lets use hstack() function from numpy to stack X_0 and X_1 horizontally (i.e. column
# This will be our final X matrix (feature matrix)
X = np.hstack((X_0, X_1, X_2, X_3))
X[:5]
```

Out[12]:
```
array([[1.        , 0.        , 3.44      , 0.44      ],
       [1.        , 0.04040404, 0.1349495 , 0.88848485],
       [1.        , 0.08080808, 0.82989899, 1.3369697 ],
       [1.        , 0.12121212, 1.52484848, 1.78545454],
       [1.        , 0.16161616, 2.21979798, 2.23393939]])
```

In [13]:
```
y = df.values[:, 3]
y[:5]
```

Out[13]:
```
array([4.38754501, 2.6796499 , 2.96848981, 3.25406475, 3.53637472])
```

In [14]:
```
theta = np.zeros(4)
theta
```

Out[14]:
```
array([0., 0., 0., 0.])
```

In [15]:
```
def compute_cost(X, y, theta):
 """
 Compute cost for linear regression.
 Input Parameters
 ----------------
 X : 2D array where each row represent the training example and each column represent
 m= number of training examples
 n= number of features (including X_0 column of ones)
 y : 1D array of labels/target value for each traing example. dimension(1 x m)
 theta : 1D array of fitting parameters or weights. Dimension (1 x n)
 Output Parameters
 -----------------
 J : Scalar value.
 """
 predictions = X.dot(theta)
 errors = np.subtract(predictions, y)
 sqrErrors = np.square(errors)
 J = 1 / (2 * m) * np.sum(sqrErrors)
 return J

def gradient_descent(X, y, theta, alpha, iterations):
    """
    Compute cost for linear regression.
    Input Parameters
    ----------------
    X : 2D array where each row represent the training example and each column represent
    m= number of training examples
    n= number of features (including X_0 column of ones)
    y : 1D array of labels/target value for each traing example. dimension(m x 1)
    theta : 1D array of fitting parameters or weights. Dimension (1 x n)
    alpha : Learning rate. Scalar value
    iterations: No of iterations. Scalar value.
    Output Parameters
    -----------------
    theta : Final Value. 1D array of fitting parameters or weights. Dimension (1 x n)
    cost_history: Conatins value of cost for each iteration. 1D array. Dimansion(m x 1)
```

```
    """
    cost_history = np.zeros(iterations)
    for i in range(iterations):
        predictions = X.dot(theta)
        errors = np.subtract(predictions, y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta;
        cost_history[i] = compute_cost(X, y, theta)
    return theta, cost_history
```

In [16]:
```
# Lets compute the cost for theta values
cost = compute_cost(X, y, theta)
print('The cost for given values of theta_0 to theta_3 =', cost)
```

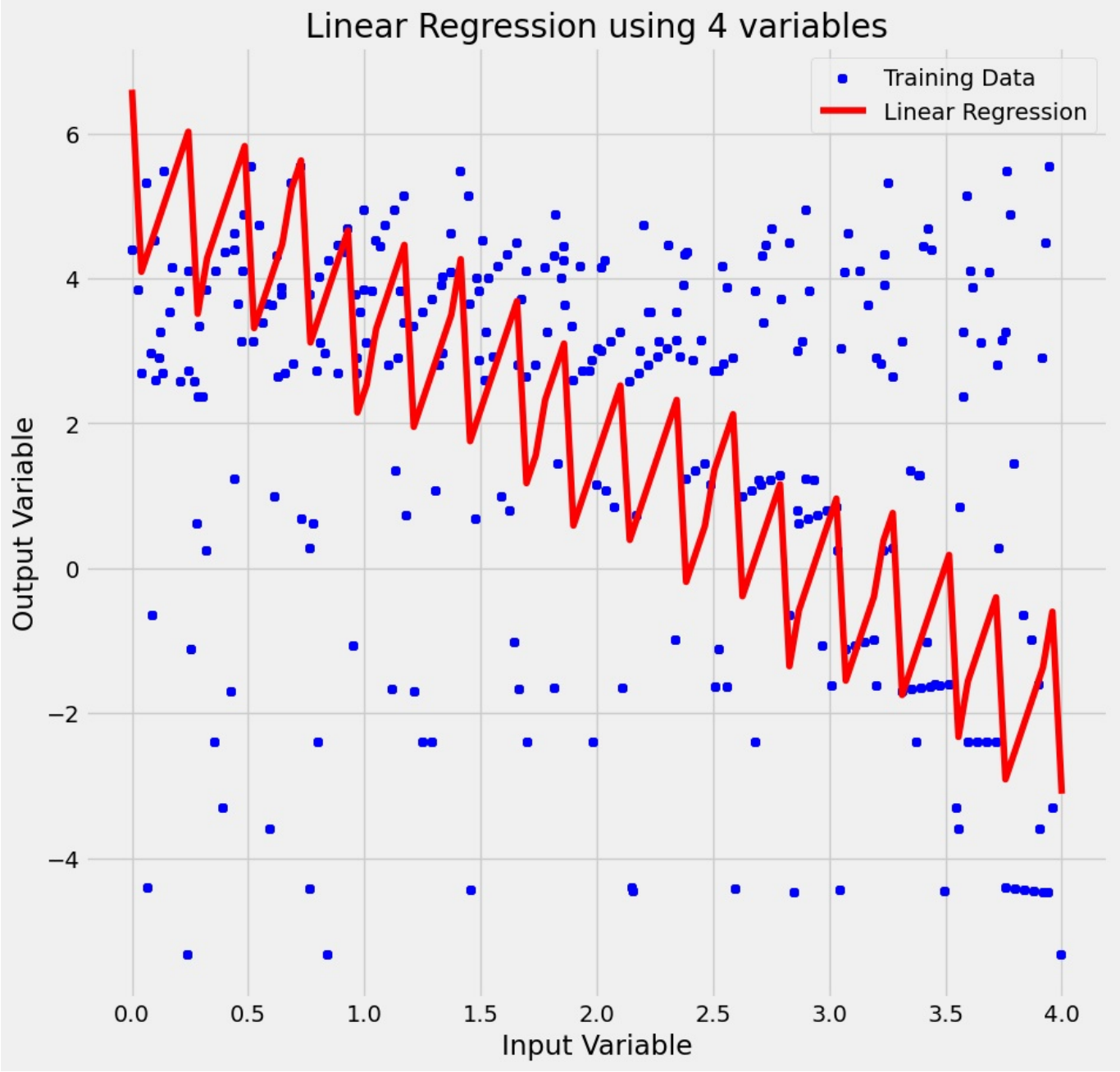
```
The cost for given values of theta_0 to theta_3 = 5.524438459196242
```

In [17]:
```
theta = [0., 0., 0., 0.]
iterations = 1500;
alpha = 0.01;

theta, cost_history = gradient_descent(X, y, theta, alpha, iterations)
print('Final value of theta =', theta)
print('cost_history =', cost_history)
```

```
Final value of theta = [ 4.15118728 -1.8394291   0.72473856 -0.09513266]
cost_history = [5.21542243 4.97171977 4.7765543  ... 0.81079843 0.81065429 0.81051044]
```

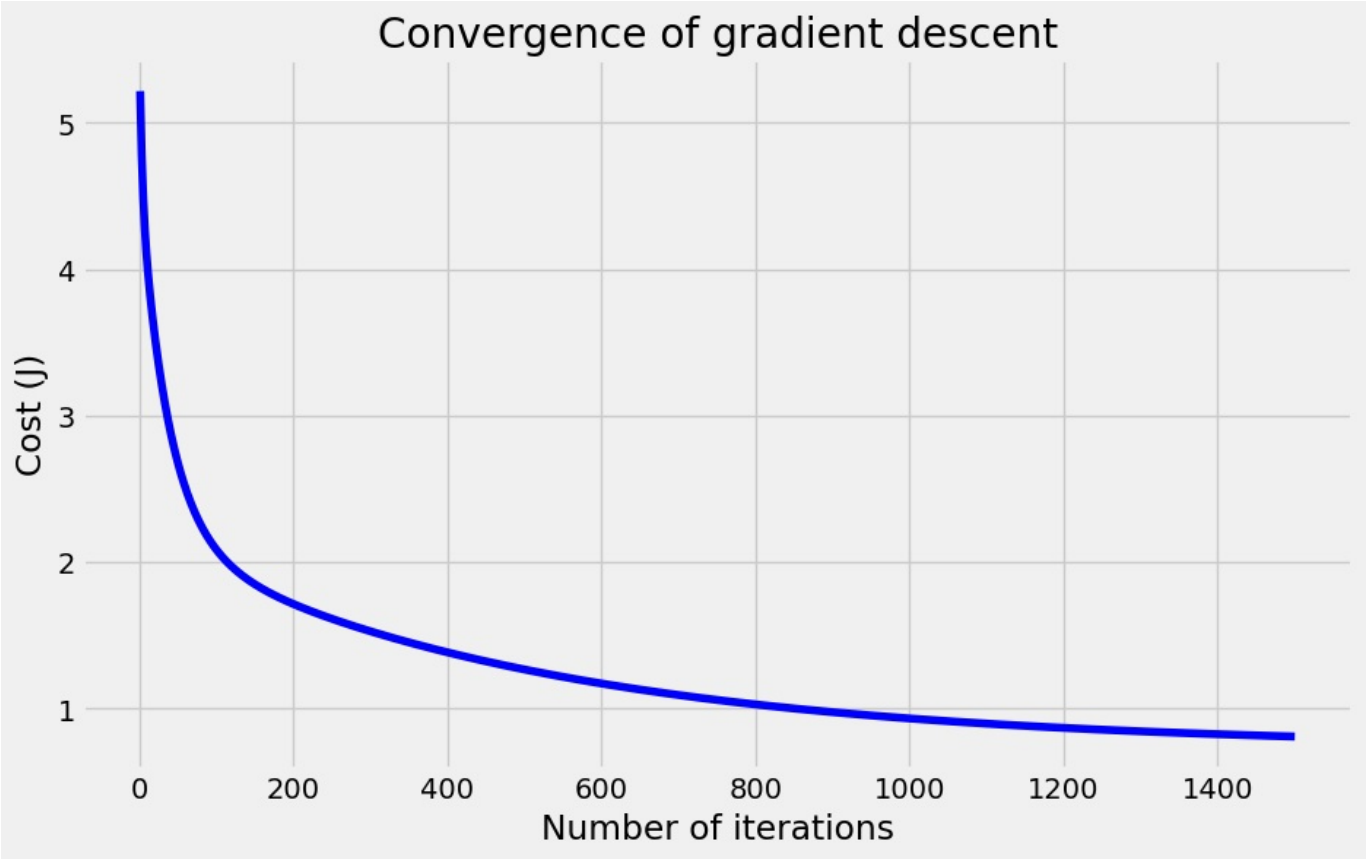
In [18]:
```
y = df.values[:, [3, 3, 3]]
plt.scatter(X[:,[1,2,3]], y, color='b', marker= '+', label= 'Training Data')
plt.plot(X[:,1],X.dot(theta), color='r', label='Linear Regression')
plt.rcParams["figure.figsize"] = (10,6)

plt.xlabel('Input Variable')
plt.ylabel('Output Variable')
plt.title('Linear Regression using 4 variables')
plt.legend()
```

Out[18]:
```
<matplotlib.legend.Legend at 0x207f0f99090>
```



In [19]:
```
plt.plot(range(1, iterations + 1),cost_history, color='blue')
plt.rcParams["figure.figsize"] = (10,6)
# plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent')
```

Out[19]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



In [20]:
```
X_0 = np.ones((3, 1))
X_0[:3]
```

Out[20]:
```
array([[1.],
       [1.],
       [1.]])
```

In [21]:
```
X_1 = ([[1], [2], [3]])
```

In [22]:
```
X_2 = ([[1], [0], [2]])
```

In [23]:
```
X_3 = ([[3], [2], [1]])
```

In [24]:
```
X = np.hstack((X_0, X_1, X_2, X_3))
X[:3]
```

Out[24]:
```
array([[1., 1., 1., 3.],
       [1., 2., 0., 2.],
       [1., 3., 2., 1.]])
```

In [25]:
```
X.dot(theta)
```

Out[25]:
```
array([ 2.75109875,  0.28206376, -0.01275555])
```

In [ ]: