X1_Regression

September 20, 2022

[18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[19]:
```
#
#df = pd.read_csv('C:\\Users\\jnguy\\Documents\\Juypter_Notebook\\HW_0\\D3.csv')
df = pd.read_csv('./D3.csv')
df.head(100)
```

[19]:
```
          X1        X2        X3         Y
0   0.000000  3.440000  0.440000  4.387545
1   0.040404  0.134949  0.888485  2.679650
2   0.080808  0.829899  1.336970  2.968490
3   0.121212  1.524848  1.785455  3.254065
4   0.161616  2.219798  2.233939  3.536375
..       …         …         …         …
95  3.838384  1.460202  3.046061 -4.440595
96  3.878788  2.155152  3.494545 -4.458663
97  3.919192  2.850101  3.943030 -4.479995
98  3.959596  3.545051  0.391515 -3.304593
99  4.000000  0.240000  0.840000 -5.332455

[100 rows x 4 columns]
```

[20]:
```
dataset = df.values[:,:]
print(dataset[:20,:])
```

```
[[0.         3.44       0.44       4.38754501]
 [0.04040404 0.1349495  0.88848485 2.6796499 ]
 [0.08080808 0.82989899 1.3369697  2.96848981]
 [0.12121212 1.52484848 1.78545454 3.25406475]
 [0.16161616 2.21979798 2.23393939 3.53637472]
 [0.2020202  2.91474747 2.68242424 3.81541972]
 [0.24242424 3.60969697 3.13090909 4.09119974]
 [0.28282828 0.30464646 3.57939394 2.36371479]
 [0.32323232 0.99959596 0.02787879 3.83296487]
 [0.36363636 1.69454546 0.47636364 4.09894997]
 [0.4040404  2.38949495 0.92484849 4.3616701 ]
```

```
 [0.44444444 3.08444444 1.37333333 4.62112526]
 [0.48484848 3.77939394 1.82181818 4.87731544]
 [0.52525252 0.47434343 2.27030303 3.13024065]
 [0.56565657 1.16929293 2.71878788 3.37990089]
 [0.60606061 1.86424242 3.16727273 3.62629616]
 [0.64646465 2.55919192 3.61575758 3.86942645]
 [0.68686869 3.25414141 0.06424242 5.30929177]
 [0.72727273 3.94909091 0.51272727 5.54589212]
 [0.76767677 0.6440404  0.96121212 3.77922749]]
```

[21]:
```
X = df.values[:,0]
Y = df.values[:,3]
len(X), len(Y)
```

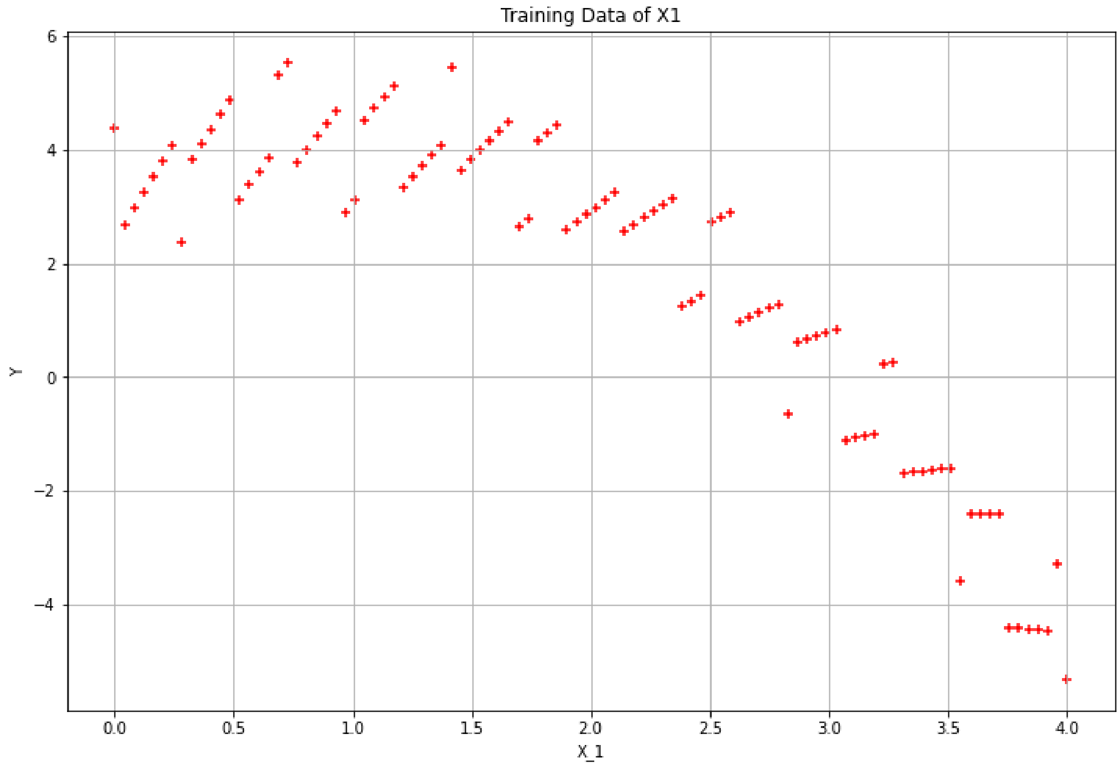
[21]:
```
(100, 100)
```

[22]:
```
print('X =', X[:5])
print('Y =', Y[:5])
```

```
X = [0.         0.04040404 0.08080808 0.12121212 0.16161616]
Y = [4.38754501 2.6796499  2.96848981 3.25406475 3.53637472]
```

[23]:
```
plt.scatter(X,Y, color='red', marker='+')
plt.grid()
plt.rcParams["figure.figsize"] = [7,7]
plt.xlabel('X_1')
plt.ylabel('Y')
plt.title('Training Data of X1')
```

[23]:
```
Text(0.5, 1.0, 'Training Data of X1')
```



[24]:
```python
# Reshape function to convert 1D array to 2D array (100x1)
m = len(X)
X_1 = X.reshape(m,1)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.        ]
 [0.04040404]
 [0.08080808]
 [0.12121212]
 [0.16161616]]
```

[25]:
```python
# Create a single column of ones (X_0)
m = len(X)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[25]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 100)
```

[26]:
```python
X = np.hstack((X_0, X_1))
X[:5]
```

[26]:
```
array([[1.        , 0.        ],
       [1.        , 0.04040404],
       [1.        , 0.08080808],
       [1.        , 0.12121212],
       [1.        , 0.16161616]])
```

[27]:
```python
theta = np.zeros((2,1))
theta
```

[27]:
```
array([[0.],
       [0.]])
```

[28]:
```python
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)

    return J
```

[29]:
```python
# Compute the cost for theta values
cost = compute_loss(X, Y, theta)
print("The cost for given theta_0 and theta_1 =", cost)
```

```
The cost for given theta_0 and theta_1 = 552.4438459196241
```

[30]:
```python
"""
Compute loss for l inear regression for all iterations

Input Parameters
```

```
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""
def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

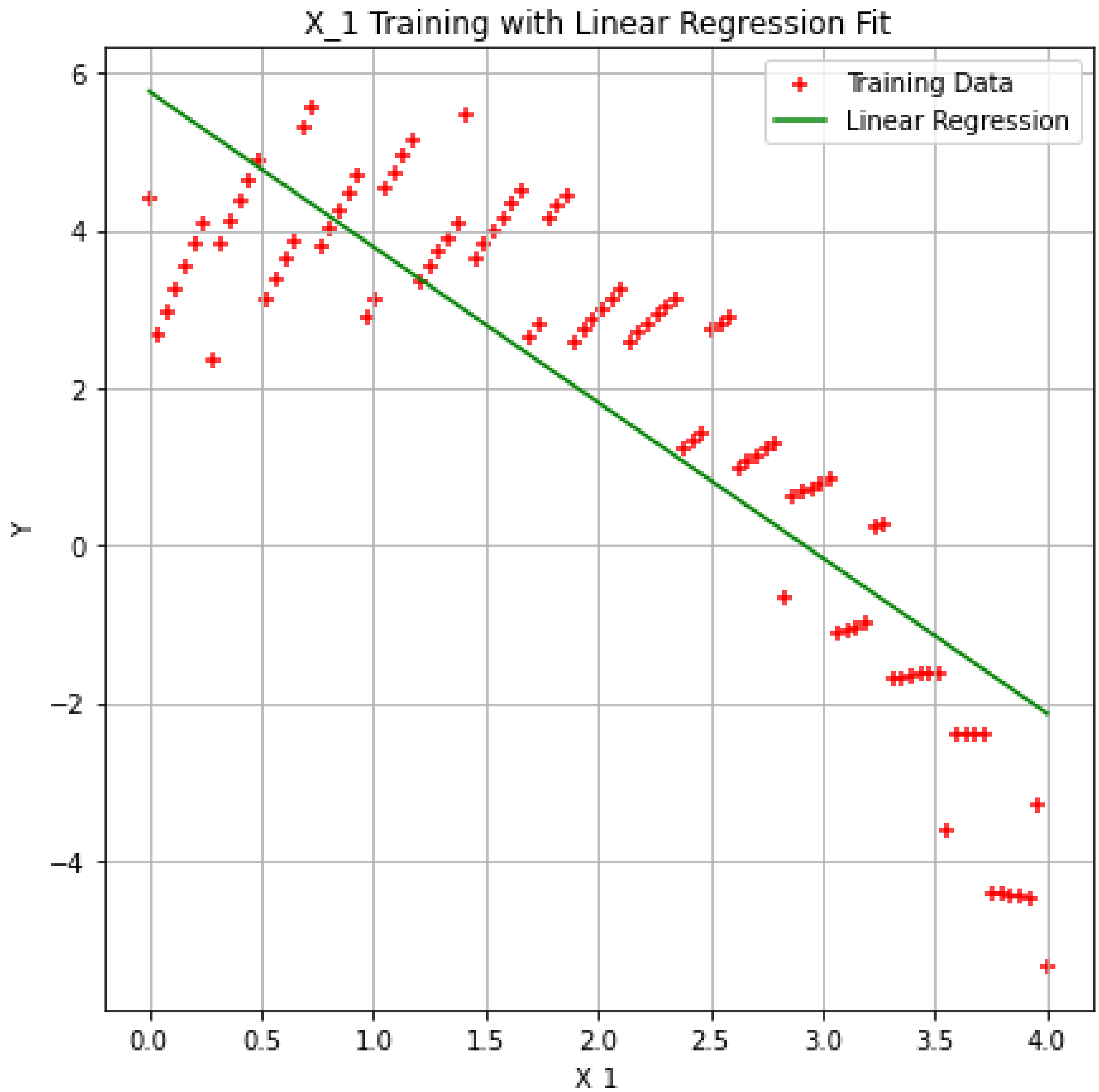
[31]:
```
theta = [0., 0.]
iterations = 1600
alpha = 0.01
```

[32]:
```
theta, loss_history = gradient_descent(X, Y, theta, alpha, iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [ 5.76017574 -1.97303847]
loss_history = [5.48226715 5.44290965 5.40604087 … 0.98861614 0.9886001
0.98858413]
```

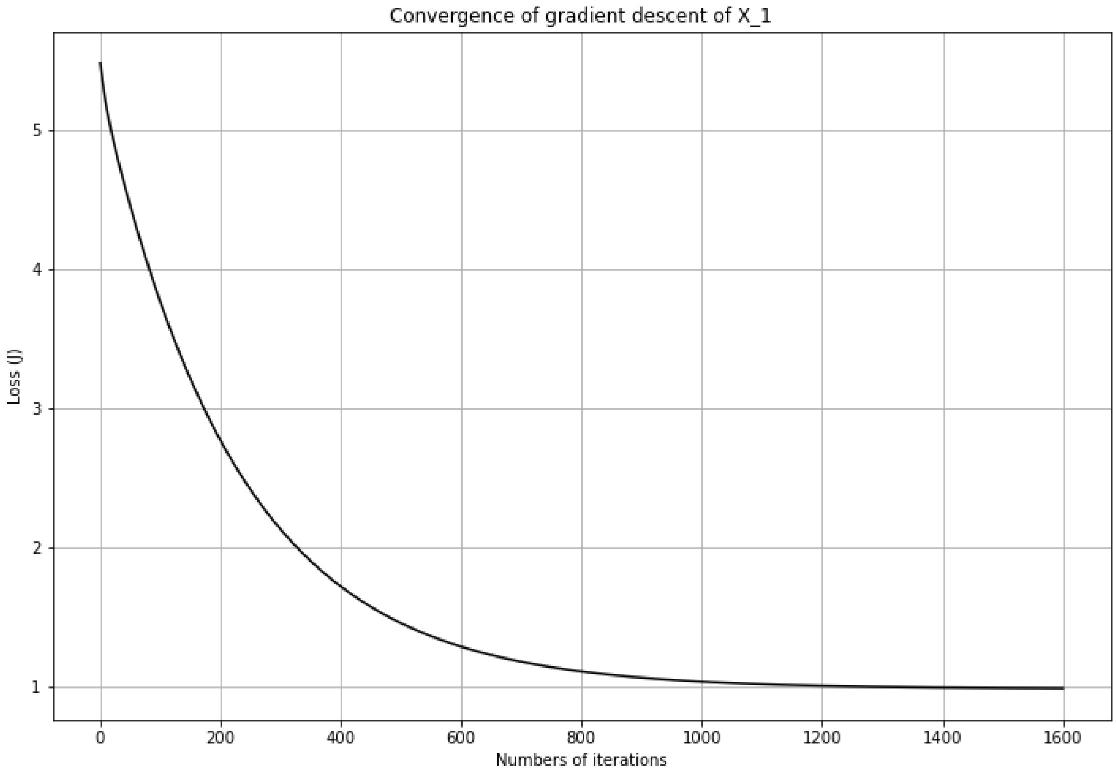
[33]:
```
# Graphing linear regression with training data points
plt.scatter(X[:,1], Y, color='red', marker='+', label= 'Training Data')
plt.plot(X[:,1], X.dot(theta), color='green', label= 'Linear Regression')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.xlabel("X_1")
plt.ylabel("Y")
plt.title("X_1 Training with Linear Regression Fit")
plt.legend()
```

[33]:
```
<matplotlib.legend.Legend at 0x1ed2f79dd30>
```



[34]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent of X_1")
```

[34]: Text(0.5, 1.0, 'Convergence of gradient descent of X_1')


Convergence of gradient descent of X_1
Loss (J)
Numbers of iterations
0
200
400
600
800
1000
1200
1400
1600
1
2
3
4
5

[ ]:

[ ]:

X2_Regression

September 20, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
#
#df = pd.read_csv('C:\\Users\\jnguy\\Documents\\Juypter_Notebook\\HW_0\\D3.csv')
df = pd.read_csv('./D3.csv')
df.head(100)
```

[2]:
```
          X1        X2        X3         Y
0   0.000000  3.440000  0.440000  4.387545
1   0.040404  0.134949  0.888485  2.679650
2   0.080808  0.829899  1.336970  2.968490
3   0.121212  1.524848  1.785455  3.254065
4   0.161616  2.219798  2.233939  3.536375
..       …         …         …         …
95  3.838384  1.460202  3.046061 -4.440595
96  3.878788  2.155152  3.494545 -4.458663
97  3.919192  2.850101  3.943030 -4.479995
98  3.959596  3.545051  0.391515 -3.304593
99  4.000000  0.240000  0.840000 -5.332455

[100 rows x 4 columns]
```

[3]:
```
dataset = df.values[:,:]
print(dataset[:20,:])
```

```
[[0.         3.44       0.44       4.38754501]
 [0.04040404 0.1349495  0.88848485 2.6796499 ]
 [0.08080808 0.82989899 1.3369697  2.96848981]
 [0.12121212 1.52484848 1.78545454 3.25406475]
 [0.16161616 2.21979798 2.23393939 3.53637472]
 [0.2020202  2.91474747 2.68242424 3.81541972]
 [0.24242424 3.60969697 3.13090909 4.09119974]
 [0.28282828 0.30464646 3.57939394 2.36371479]
 [0.32323232 0.99959596 0.02787879 3.83296487]
 [0.36363636 1.69454546 0.47636364 4.09894997]
 [0.4040404  2.38949495 0.92484849 4.3616701 ]
```

```
 [0.44444444 3.08444444 1.37333333 4.62112526]
 [0.48484848 3.77939394 1.82181818 4.87731544]
 [0.52525252 0.47434343 2.27030303 3.13024065]
 [0.56565657 1.16929293 2.71878788 3.37990089]
 [0.60606061 1.86424242 3.16727273 3.62629616]
 [0.64646465 2.55919192 3.61575758 3.86942645]
 [0.68686869 3.25414141 0.06424242 5.30929177]
 [0.72727273 3.94909091 0.51272727 5.54589212]
 [0.76767677 0.6440404  0.96121212 3.77922749]]
```

[4]:
```
X = df.values[:,1]
Y = df.values[:,3]
len(X), len(Y)
```

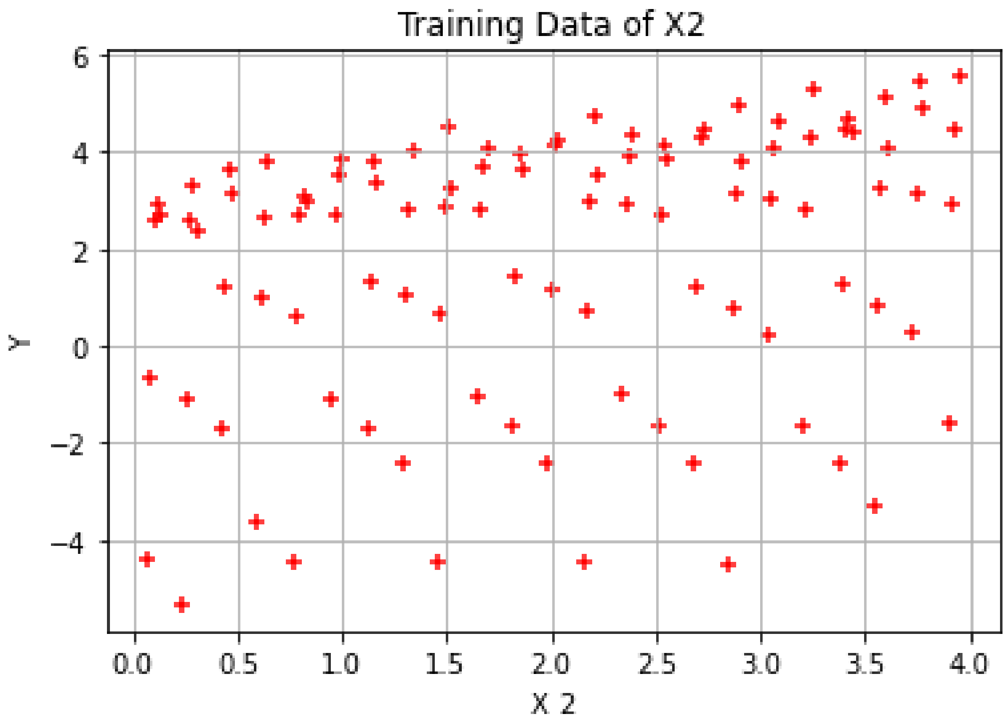
[4]:
```
(100, 100)
```

[5]:
```
print('X =', X[:5])
print('Y =', Y[:5])
```

```
X = [3.44       0.1349495  0.82989899 1.52484848 2.21979798]
Y = [4.38754501 2.6796499  2.96848981 3.25406475 3.53637472]
```

[6]:
```
plt.scatter(X,Y, color='red', marker='+')
plt.grid()
plt.rcParams["figure.figsize"] = [7,7]
plt.xlabel('X_2')
plt.ylabel('Y')
plt.title('Training Data of X2')
```

[6]:
```
Text(0.5, 1.0, 'Training Data of X2')
```



[7]:
```python
# Reshape function to convert 1D array to 2D array (100x1)
m = len(X)
X_1 = X.reshape(m,1)
print("X_2 =", X_1[:5,:])
```

```
X_2 = [[3.44      ]
 [0.1349495 ]
 [0.82989899]
 [1.52484848]
 [2.21979798]]
```

[8]:
```python
# Create a single column of ones (X_0)
m = len(X)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[8]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 100)
```

[9]:
```
X = np.hstack((X_0, X_1))
X[:5]
```

[9]:
```
array([[1.        , 3.44      ],
       [1.        , 0.1349495 ],
       [1.        , 0.82989899],
       [1.        , 1.52484848],
       [1.        , 2.21979798]])
```

[10]:
```
theta = np.zeros((2,1))
theta
```

[10]:
```
array([[0.],
       [0.]])
```

[11]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)

    return J
```

[12]:
```
# Compute the cost for theta values
cost = compute_loss(X, Y, theta)
print("The cost for given theta_0 and theta_2 =", cost)
```

```
The cost for given theta_0 and theta_2 = 552.4438459196241
```

[13]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
```

```
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""
def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y) # Error (m,1) = temp
        sum_delta = (alpha / m) * X.transpose().dot(errors); # sum_delta (n,1)
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[14]:
```
theta = [0., 0.]
iterations = 500
alpha = 0.004
```

[15]:
```
theta, loss_history = gradient_descent(X, Y, theta, alpha, iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

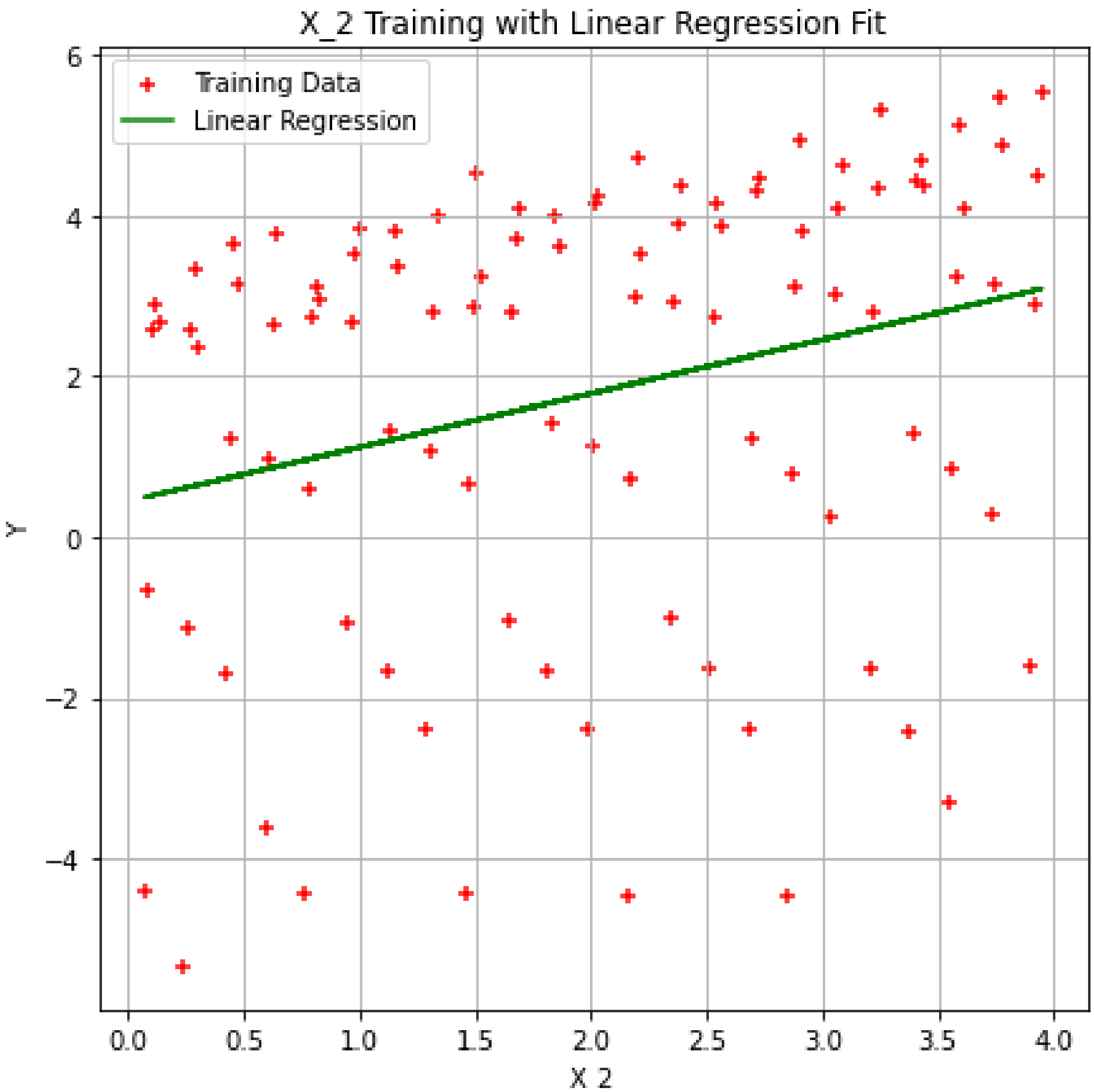
```
Final value of theta = [0.44673177 0.67021365]
loss_history = [5.43227213 5.34457428 5.26112817 5.18172756 5.10617622
5.03428746
 4.9658836  4.90079563 4.83886269 4.77993177 4.72385723 4.67050055
 4.61972988 4.57141979 4.52545094 4.48170977 4.44008822 4.40048347
 4.36279772 4.32693787 4.29281537 4.26034595 4.22944942 4.2000495
 4.17207359 4.14545262 4.12012088 4.09601583 4.07307798 4.05125071
 4.03048016 4.01071507 3.99190667 3.97400857 3.95697661 3.94076878
 3.92534512 3.91066758 3.89669998 3.88340788 3.87075854 3.85872076
 3.84726489 3.83636271 3.82598736 3.81611329 3.8067162  3.79777294
 3.78926151 3.78116096 3.77345137 3.76611378 3.75913015 3.75248331
 3.74615692 3.74013546 3.73440412 3.72894885 3.72375625 3.71881358
 3.71410873 3.70963017 3.70536692 3.70130853 3.69744509 3.69376713
 3.69026565 3.68693212 3.68375837 3.68073667 3.67785964 3.67512026
 3.67251187 3.67002812 3.66766295 3.66541063 3.66326568 3.6612229
 3.65927733 3.65742427 3.65565922 3.65397793 3.65237633 3.65085056
 3.64939695 3.648012   3.64669238 3.64543493 3.64423663 3.64309462
```

3.64200617	3.64096869	3.63997971	3.63903687	3.63813795	3.63728081
3.63646344	3.6356839	3.63494037	3.6342311	3.63355444	3.6329088
3.63229268	3.63170466	3.63114337	3.63060752	3.63009588	3.62960728
3.62914061	3.6286948	3.62826885	3.62786179	3.62747271	3.62710075
3.62674507	3.62640489	3.62607946	3.62576807	3.62547004	3.62518472
3.62491151	3.62464981	3.62439908	3.62415878	3.62392841	3.62370749
3.62349557	3.62329221	3.623097	3.62290955	3.62272948	3.62255645
3.6223901	3.62223013	3.62207622	3.62192808	3.62178544	3.62164803
3.62151561	3.62138793	3.62126476	3.62114589	3.62103112	3.62092025
3.62081308	3.62070946	3.6206092	3.62051215	3.62041815	3.62032706
3.62023874	3.62015306	3.62006989	3.61998912	3.61991063	3.61983431
3.61976006	3.61968779	3.6196174	3.6195488	3.61948191	3.61941665
3.61935294	3.61929072	3.6192299	3.61917042	3.61911223	3.61905527
3.61899947	3.61894478	3.61889116	3.61883855	3.61878691	3.61873618
3.61868635	3.61863735	3.61858915	3.61854172	3.61849503	3.61844903
3.6184037	3.61835901	3.61831493	3.61827144	3.6182285	3.6181861
3.61814421	3.61810281	3.61806187	3.61802138	3.61798132	3.61794168
3.61790242	3.61786354	3.61782503	3.61778686	3.61774902	3.61771151
3.6176743	3.61763738	3.61760074	3.61756438	3.61752828	3.61749243
3.61745682	3.61742144	3.61738629	3.61735135	3.61731662	3.61728209
3.61724776	3.61721361	3.61717964	3.61714584	3.61711222	3.61707875
3.61704545	3.61701229	3.61697928	3.61694642	3.61691369	3.6168811
3.61684864	3.61681631	3.61678409	3.616752	3.61672003	3.61668817
3.61665642	3.61662477	3.61659323	3.6165618	3.61653046	3.61649922
3.61646808	3.61643703	3.61640607	3.6163752	3.61634442	3.61631372
3.61628311	3.61625258	3.61622213	3.61619176	3.61616147	3.61613126
3.61610112	3.61607106	3.61604107	3.61601115	3.6159813	3.61595152
3.61592181	3.61589217	3.6158626	3.61583309	3.61580365	3.61577428
3.61574497	3.61571572	3.61568653	3.61565741	3.61562835	3.61559935
3.61557041	3.61554153	3.61551271	3.61548395	3.61545524	3.6154266
3.61539801	3.61536948	3.615341	3.61531258	3.61528422	3.61525591
3.61522766	3.61519946	3.61517132	3.61514323	3.6151152	3.61508722
3.61505929	3.61503141	3.61500359	3.61497582	3.6149481	3.61492044
3.61489282	3.61486526	3.61483775	3.61481029	3.61478288	3.61475552
3.61472821	3.61470096	3.61467375	3.61464659	3.61461948	3.61459243
3.61456542	3.61453846	3.61451155	3.61448469	3.61445787	3.61443111
3.6144044	3.61437773	3.61435111	3.61432454	3.61429802	3.61427154
3.61424512	3.61421874	3.6141924	3.61416612	3.61413988	3.61411369
3.61408755	3.61406145	3.61403541	3.6140094	3.61398345	3.61395754
3.61393167	3.61390586	3.61388009	3.61385436	3.61382868	3.61380305
3.61377746	3.61375192	3.61372643	3.61370097	3.61367557	3.61365021
3.6136249	3.61359963	3.6135744	3.61354922	3.61352409	3.613499
3.61347395	3.61344895	3.61342399	3.61339908	3.61337421	3.61334939
3.61332461	3.61329988	3.61327519	3.61325054	3.61322593	3.61320137
3.61317686	3.61315238	3.61312796	3.61310357	3.61307923	3.61305493
3.61303067	3.61300646	3.61298229	3.61295816	3.61293407	3.61291003
3.61288603	3.61286208	3.61283816	3.61281429	3.61279046	3.61276667
3.61274293	3.61271923	3.61269557	3.61267195	3.61264837	3.61262483

```
3.61260134 3.61257789 3.61255448 3.61253111 3.61250778 3.6124845
3.61246125 3.61243805 3.61241489 3.61239177 3.61236869 3.61234565
3.61232265 3.61229969 3.61227677 3.6122539  3.61223106 3.61220827
3.61218551 3.6121628  3.61214012 3.61211749 3.61209489 3.61207234
3.61204982 3.61202735 3.61200492 3.61198252 3.61196017 3.61193785
3.61191558 3.61189334 3.61187114 3.61184898 3.61182687 3.61180479
3.61178275 3.61176075 3.61173878 3.61171686 3.61169498 3.61167313
3.61165133 3.61162956 3.61160783 3.61158614 3.61156449 3.61154287
3.6115213  3.61149976 3.61147826 3.6114568  3.61143537 3.61141399
3.61139264 3.61137133 3.61135006 3.61132883 3.61130763 3.61128647
3.61126535 3.61124426 3.61122322 3.61120221 3.61118124 3.6111603
3.6111394  3.61111854 3.61109772 3.61107693 3.61105618 3.61103547
3.61101479 3.61099415 3.61097355 3.61095298 3.61093245 3.61091196
3.6108915  3.61087108 3.61085069 3.61083034 3.61081003 3.61078975
3.61076951 3.6107493  3.61072913 3.610709   3.6106889  3.61066884
3.61064881 3.61062882 3.61060886 3.61058894 3.61056906 3.61054921
3.61052939 3.61050961 3.61048987 3.61047016 3.61045048 3.61043084
3.61041124 3.61039167 3.61037213 3.61035263 3.61033316 3.61031373
3.61029433 3.61027497 3.61025564 3.61023634 3.61021708 3.61019786
3.61017866 3.61015951 3.61014038 3.61012129 3.61010223 3.61008321
3.61006422 3.61004527]
```

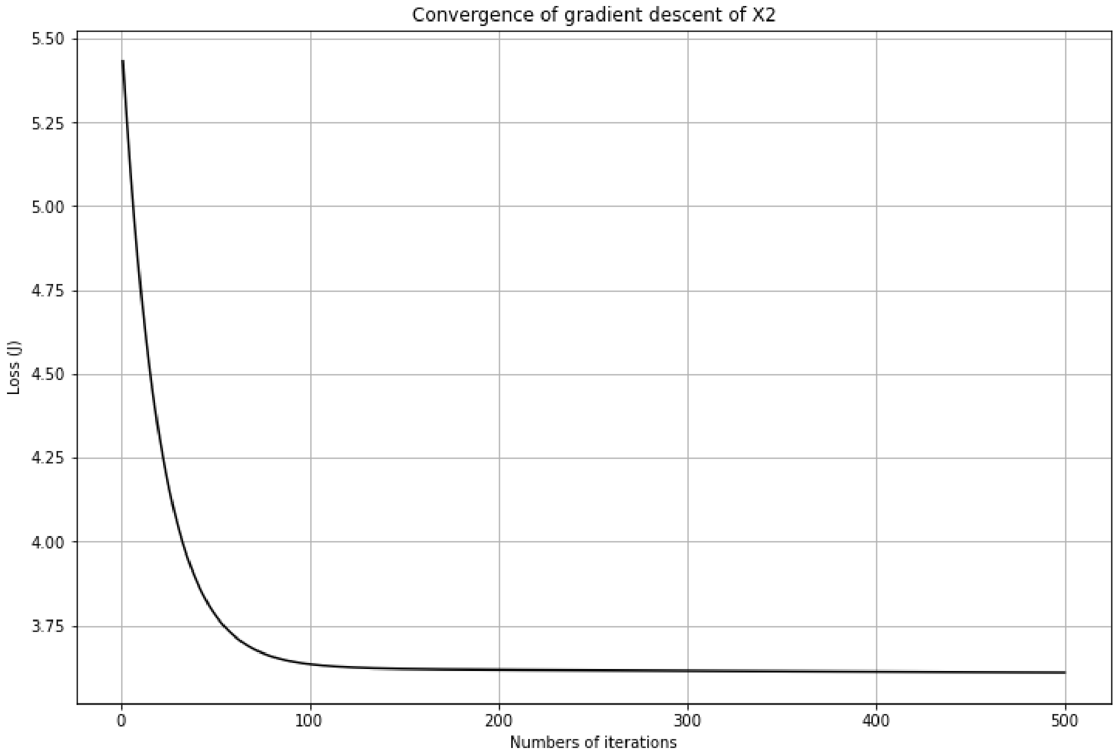
[16]:
```python
# Graphing linear regression with training data points
plt.scatter(X[:,1], Y, color='red', marker='+', label= 'Training Data')
plt.plot(X[:,1], X.dot(theta), color='green', label= 'Linear Regression')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.xlabel("X_2")
plt.ylabel("Y")
plt.title("X_2 Training with Linear Regression Fit")
plt.legend()
```

[16]:
```
<matplotlib.legend.Legend at 0x1e3c5487b80>
```



[17]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent of X2")
```

[17]: Text(0.5, 1.0, 'Convergence of gradient descent of X2')


Convergence of gradient descent of X2
5.50
5.25
5.00
4.75
4.50
4.25
4.00
3.75
Loss (J)
0
100
200
300
400
500
Numbers of iterations

[ ]:

[ ]:

X3_Regression

September 20, 2022

[24]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[25]:
```
#
df = pd.read_csv('https://github.com/Norumai01/Intro_Machine_Learning/raw/main/
↪HW_0/D3.csv')
#df = pd.read_csv('./D3.csv')
df.head(100)
```

[25]:
```
          X1        X2        X3         Y
0   0.000000  3.440000  0.440000  4.387545
1   0.040404  0.134949  0.888485  2.679650
2   0.080808  0.829899  1.336970  2.968490
3   0.121212  1.524848  1.785455  3.254065
4   0.161616  2.219798  2.233939  3.536375
..       …         …         …        …
95  3.838384  1.460202  3.046061 -4.440595
96  3.878788  2.155152  3.494545 -4.458663
97  3.919192  2.850101  3.943030 -4.479995
98  3.959596  3.545051  0.391515 -3.304593
99  4.000000  0.240000  0.840000 -5.332455

[100 rows x 4 columns]
```

[26]:
```
dataset = df.values[:,:]
print(dataset[:20,:])
```

```
[[0.         3.44       0.44       4.38754501]
 [0.04040404 0.1349495  0.88848485 2.6796499 ]
 [0.08080808 0.82989899 1.3369697  2.96848981]
 [0.12121212 1.52484848 1.78545454 3.25406475]
 [0.16161616 2.21979798 2.23393939 3.53637472]
 [0.2020202  2.91474747 2.68242424 3.81541972]
 [0.24242424 3.60969697 3.13090909 4.09119974]
 [0.28282828 0.30464646 3.57939394 2.36371479]
 [0.32323232 0.99959596 0.02787879 3.83296487]
 [0.36363636 1.69454546 0.47636364 4.09894997]
```

```
 [0.4040404  2.38949495 0.92484849 4.3616701 ]
 [0.44444444 3.08444444 1.37333333 4.62112526]
 [0.48484848 3.77939394 1.82181818 4.87731544]
 [0.52525252 0.47434343 2.27030303 3.13024065]
 [0.56565657 1.16929293 2.71878788 3.37990089]
 [0.60606061 1.86424242 3.16727273 3.62629616]
 [0.64646465 2.55919192 3.61575758 3.86942645]
 [0.68686869 3.25414141 0.06424242 5.30929177]
 [0.72727273 3.94909091 0.51272727 5.54589212]
 [0.76767677 0.6440404  0.96121212 3.77922749]]
```

[27]:
```
X = df.values[:,2]
Y = df.values[:,3]
len(X), len(Y)
```

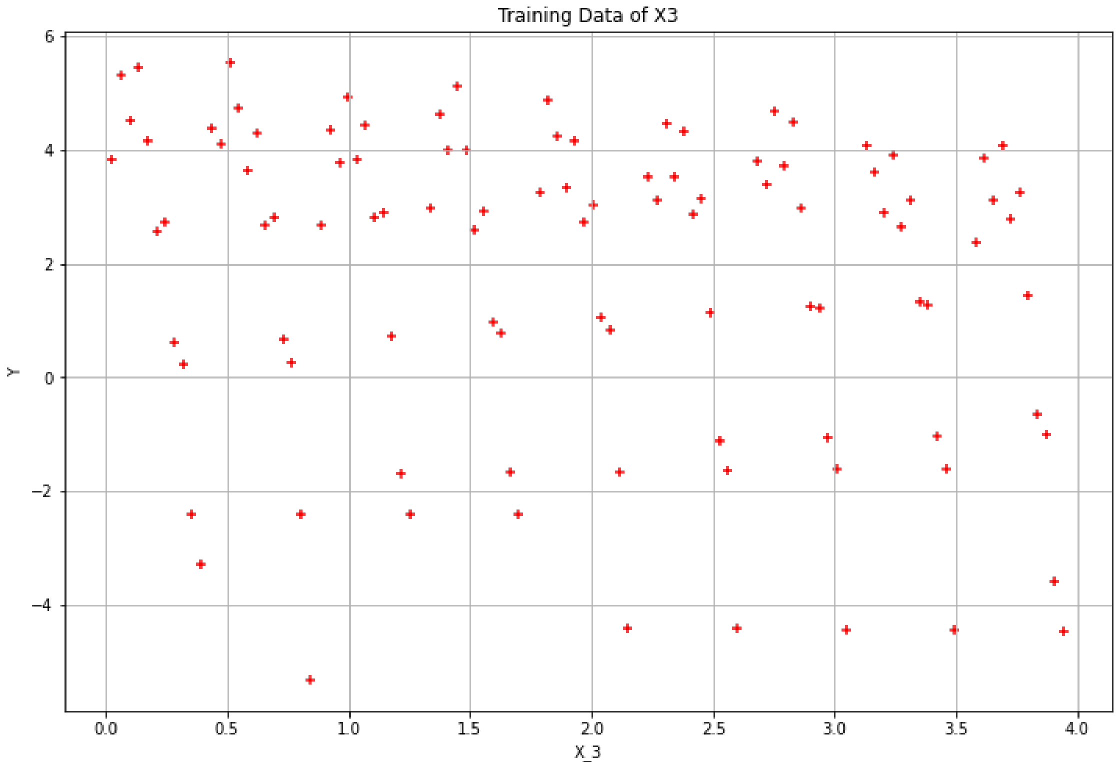
[27]:
```
(100, 100)
```

[28]:
```
print('X =', X[:5])
print('Y =', Y[:5])
```

```
X = [0.44       0.88848485 1.3369697  1.78545454 2.23393939]
Y = [4.38754501 2.6796499  2.96848981 3.25406475 3.53637472]
```

[29]:
```
plt.scatter(X,Y, color='red', marker='+')
plt.grid()
plt.rcParams["figure.figsize"] = [7,7]
plt.xlabel('X_3')
plt.ylabel('Y')
plt.title('Training Data of X3')
```

[29]:
```
Text(0.5, 1.0, 'Training Data of X3')
```



[30]:
```
# Reshape function to convert 1D array to 2D array (100x1)
m = len(X)
X_1 = X.reshape(m,1)
print("X_3 =", X_1[:5,:])
```

```
X_3 = [[0.44      ]
 [0.88848485]
 [1.3369697 ]
 [1.78545454]
 [2.23393939]]
```

[31]:
```
# Create a single column of ones (X_0)
m = len(X)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[31]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 100)
```

[32]:
```python
X = np.hstack((X_0, X_1))
X[:5]
```

[32]:
```
array([[1.        , 0.44      ],
       [1.        , 0.88848485],
       [1.        , 1.3369697 ],
       [1.        , 1.78545454],
       [1.        , 2.23393939]])
```

[33]:
```python
theta = np.zeros((2,1))
theta
```

[33]:
```
array([[0.],
       [0.]])
```

[34]:
```python
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)

    return J
```

[35]:
```python
# Compute the cost for theta values
cost = compute_loss(X, Y, theta)
print("The cost for given theta_0 and theta_3 =", cost)
```

```
The cost for given theta_0 and theta_3 = 552.4438459196241
```

[36]:
```python
"""
Compute loss for l inear regression for all iterations

Input Parameters
```

```
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""
def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

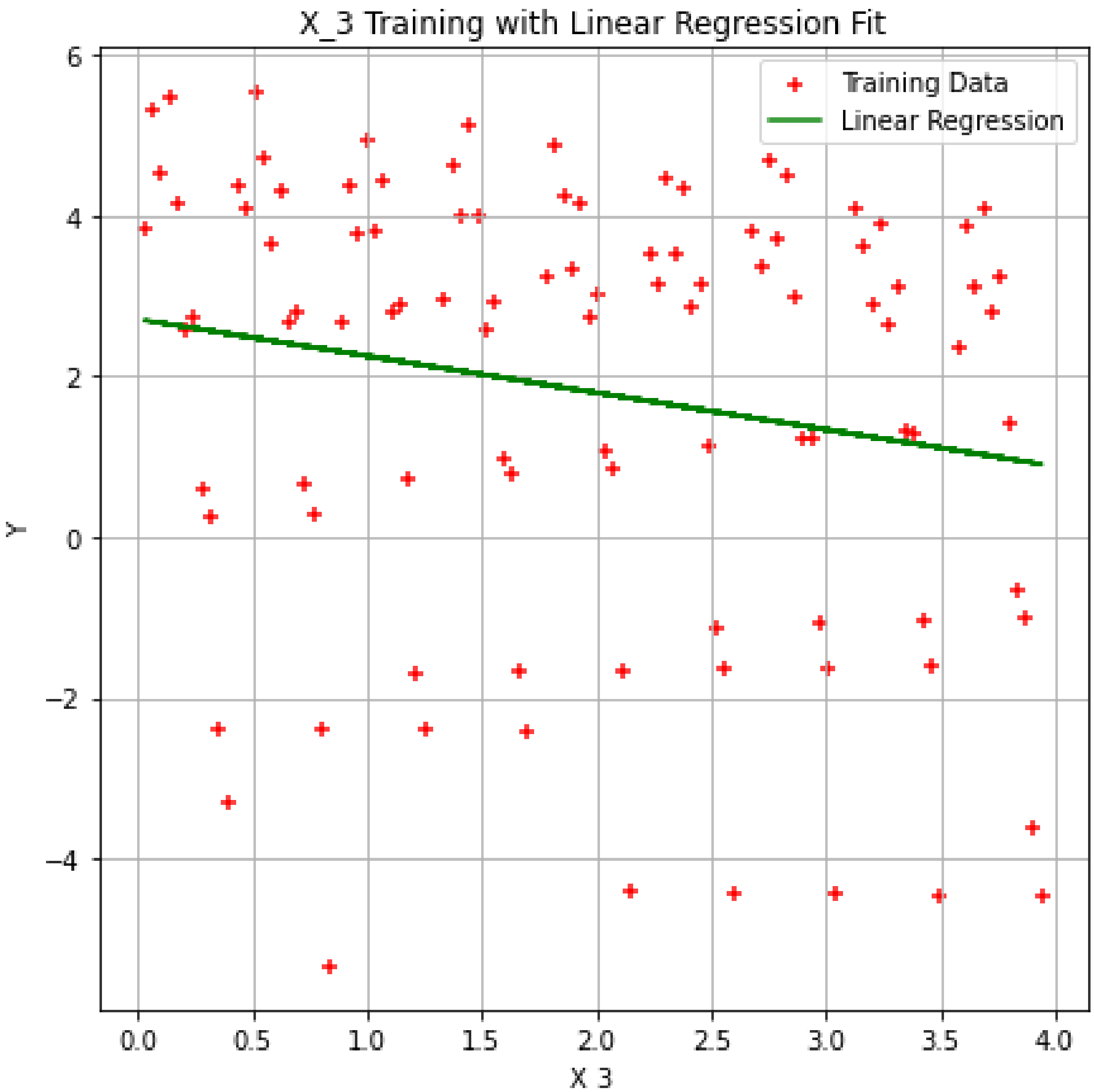
[37]:
```
theta = [0., 0.]
iterations = 1250
alpha = 0.01
```

[38]:
```
theta, loss_history = gradient_descent(X, Y, theta, alpha, iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [ 2.71178238 -0.45734628]
loss_history = [5.40768785 5.30397076 5.21178297 … 3.63279409 3.63277908
3.63276413]
```

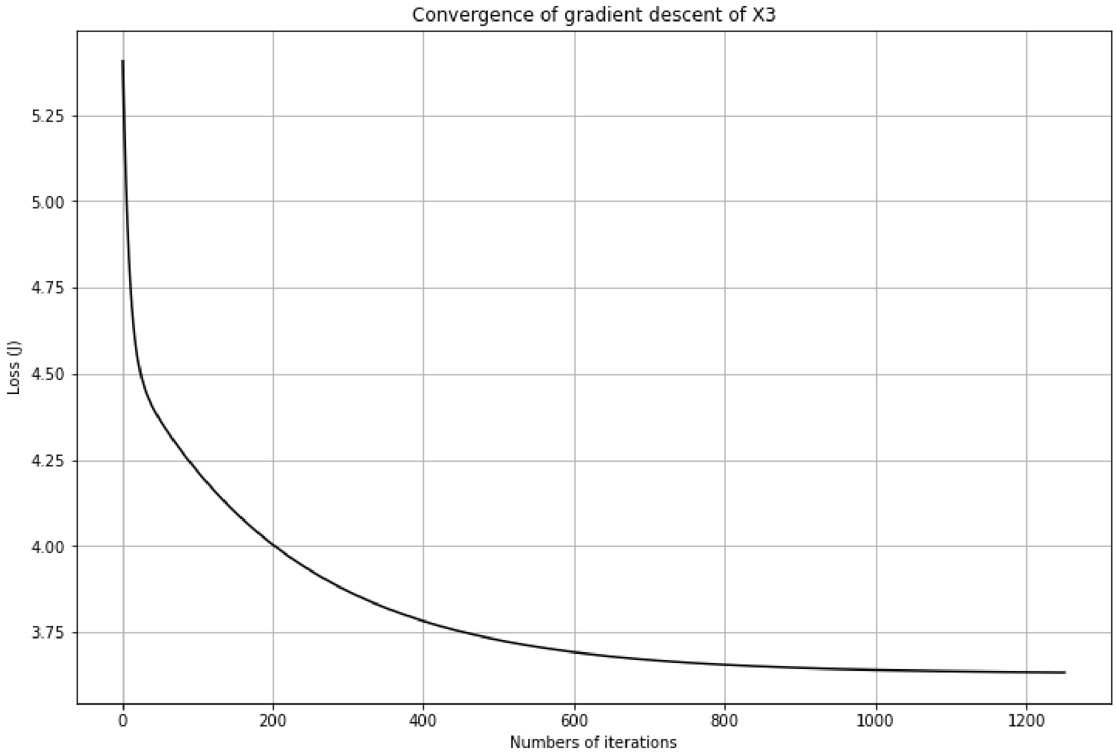
[39]:
```
# Graphing linear regression with training data points
plt.scatter(X[:,1], Y, color='red', marker='+', label= 'Training Data')
plt.plot(X[:,1], X.dot(theta), color='green', label= 'Linear Regression')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.xlabel("X_3")
plt.ylabel("Y")
plt.title("X_3 Training with Linear Regression Fit")
plt.legend()
```

[39]: <matplotlib.legend.Legend at 0x27264e68c40>




```
[40]: plt.plot(range(1, iterations + 1), loss_history, color = 'black')
      plt.rcParams["figure.figsize"] = [12,8]
      plt.grid()
      plt.xlabel("Numbers of iterations")
      plt.ylabel("Loss (J)")
      plt.title("Convergence of gradient descent of X3")
```

```
[40]: Text(0.5, 1.0, 'Convergence of gradient descent of X3')
```


Convergence of gradient descent of X3
Loss (J)
5.25
5.00
4.75
4.50
4.25
4.00
3.75
0
200
400
600
800
1000
1200
Numbers of iterations

[ ]:

[ ]:

Multivariable_Regression

September 20, 2022

[96]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[97]:
```
#
df = pd.read_csv('./D3.csv')
df.head(100)
```

[97]:
```
          X1        X2        X3         Y
0   0.000000  3.440000  0.440000  4.387545
1   0.040404  0.134949  0.888485  2.679650
2   0.080808  0.829899  1.336970  2.968490
3   0.121212  1.524848  1.785455  3.254065
4   0.161616  2.219798  2.233939  3.536375
..       …         …         …         …
95  3.838384  1.460202  3.046061 -4.440595
96  3.878788  2.155152  3.494545 -4.458663
97  3.919192  2.850101  3.943030 -4.479995
98  3.959596  3.545051  0.391515 -3.304593
99  4.000000  0.240000  0.840000 -5.332455

[100 rows x 4 columns]
```

[98]:
```
dataset = df.values[:,:]
print(dataset[:20,:])
```

```
[[0.         3.44       0.44       4.38754501]
 [0.04040404 0.1349495  0.88848485 2.6796499 ]
 [0.08080808 0.82989899 1.3369697  2.96848981]
 [0.12121212 1.52484848 1.78545454 3.25406475]
 [0.16161616 2.21979798 2.23393939 3.53637472]
 [0.2020202  2.91474747 2.68242424 3.81541972]
 [0.24242424 3.60969697 3.13090909 4.09119974]
 [0.28282828 0.30464646 3.57939394 2.36371479]
 [0.32323232 0.99959596 0.02787879 3.83296487]
 [0.36363636 1.69454546 0.47636364 4.09894997]
 [0.4040404  2.38949495 0.92484849 4.3616701 ]
 [0.44444444 3.08444444 1.37333333 4.62112526]
```

```
 [0.48484848 3.77939394 1.82181818 4.87731544]
 [0.52525252 0.47434343 2.27030303 3.13024065]
 [0.56565657 1.16929293 2.71878788 3.37990089]
 [0.60606061 1.86424242 3.16727273 3.62629616]
 [0.64646465 2.55919192 3.61575758 3.86942645]
 [0.68686869 3.25414141 0.06424242 5.30929177]
 [0.72727273 3.94909091 0.51272727 5.54589212]
 [0.76767677 0.6440404  0.96121212 3.77922749]]
```

[99]:
```
X = df.values[:,0:3]
Y = df.values[:,3]
len(X), len(Y)
```

[99]:
```
(100, 100)
```

[100]:
```
print('X =', X[:5])
print('Y =', Y[:5])
```

```
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 2.6796499  2.96848981 3.25406475 3.53637472]
```

[ ]:

[101]:
```
# Reshape function to convert 1D array to 2D array (100x1)
m = len(X)
X_1 = X.reshape(m,3)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[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]]
```

[102]:
```
# Create a single column of ones (X_0)
m = len(X)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[102]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 100)
```

[103]:
```
X = np.hstack((X_0, X_1))
X[:5]
```

[103]:
```
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]])
```

[104]:
```
theta = np.zeros((4,1))
theta
```

[104]:
```
array([[0.],
       [0.],
       [0.],
       [0.]])
```

[105]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)

    return J
```

[106]:
```
# Compute the cost for theta values
cost = compute_loss(X, Y, theta)
print("The cost for all given theta =", cost)
```

```
The cost for all given theta = 552.4438459196241
```

[107]:
```
"""
Compute loss for l inear regression for all iterations
```

```
Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""
def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[108]:
```
theta = [0., 0., 0., 0.]
iterations = 2500
alpha = 0.01
```

[109]:
```
theta, loss_history = gradient_descent(X, Y, theta, alpha, iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

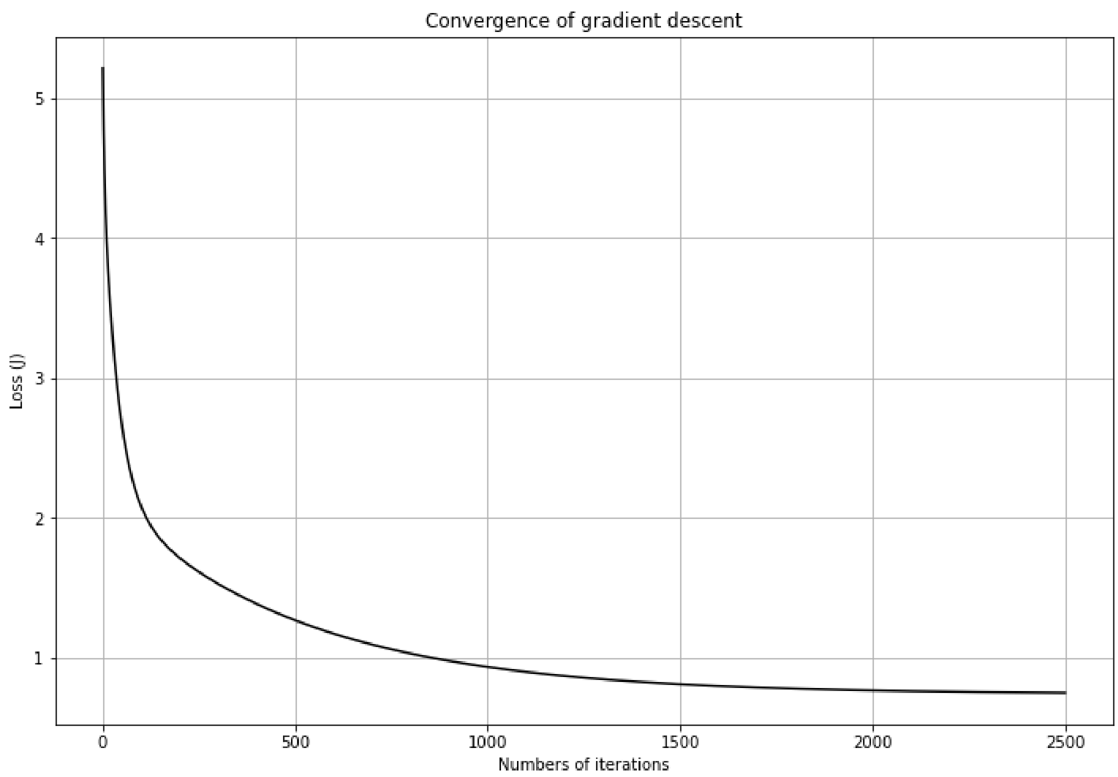
```
Final value of theta = [ 4.88518623 -1.94311861  0.60344978 -0.20272198]
loss_history = [5.21542243 4.97171977 4.7765543  … 0.74830606 0.74828645
0.74826687]
```

[ ]:
```

```

[110]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[110]: Text(0.5, 1.0, 'Convergence of gradient descent')



[111]:
```
print(theta)
```

```
[ 4.88518623 -1.94311861  0.60344978 -0.20272198]
```

[112]:
```
theta0 = theta[0]
theta1 = theta[1]
theta2 = theta[2]
theta3 = theta[3]
new_x1 = 1
new_x2 = 1
new_x3 = 1
# X0 = 1
prediction1 = new_x3*theta3 + new_x2*theta2 + new_x1*theta1 + theta0*1
print("The new Y value of h(1,1,1) is",prediction1)
```

```
The new Y value of h(1,1,1) is 3.34279542031987
```

[113]:
```
theta0 = theta[0]
theta1 = theta[1]
theta2 = theta[2]
theta3 = theta[3]
new_x1 = 2
new_x2 = 0
```

```
new_x3 = 4
# X0 = 1
prediction2 = new_x3*theta3 + new_x2*theta2 + new_x1*theta1 + theta0*1
print("The new Y value of h(2,0,4) is",prediction2)
```

```
The new Y value of h(2,0,4) is 0.18806109519189196
```

[114]:
```
theta0 = theta[0]
theta1 = theta[1]
theta2 = theta[2]
theta3 = theta[3]
new_x1 = 3
new_x2 = 2
new_x3 = 1
# X0 = 1
prediction3 = new_x3*theta3 + new_x2*theta2 + new_x1*theta1 + theta0*1
print("The new Y value of h(3,2,1) is",prediction3)
```

```
The new Y value of h(3,2,1) is 0.06000797936338653
```

[ ]:
```

```