Linear Regression

import library

In [1]:

```
import numpy as np
import matplotlib.image as img
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from mpl_toolkits.mplot3d import Axes3D
```

load point data for training and testing

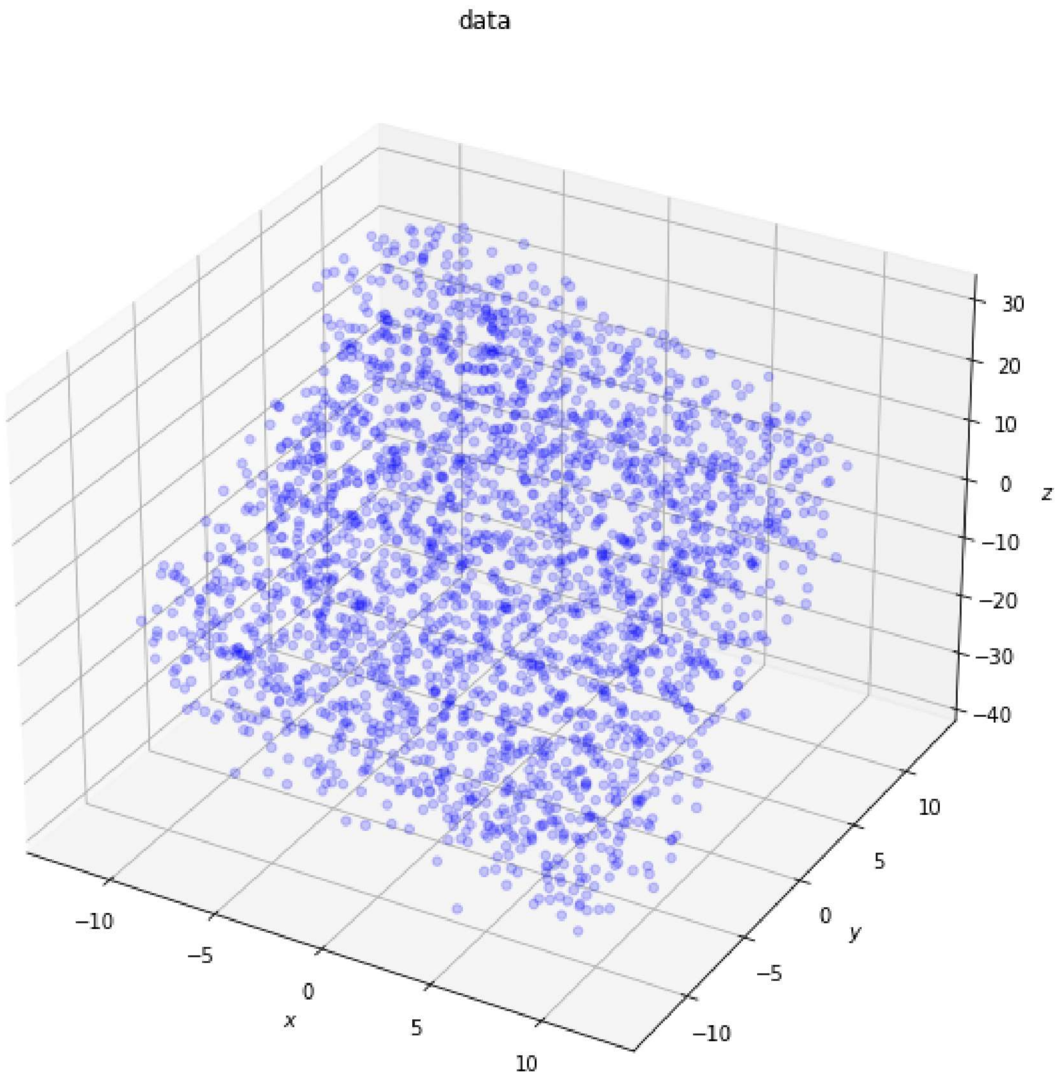
In [2]:

```
fname_data  = 'assignment_07_data.csv'

data        = np.genfromtxt(fname_data, delimiter=',')
num_data    = data.shape[0]

xx          = np.zeros(num_data)
yy          = np.zeros(num_data)
zz          = np.zeros(num_data)

for i in range(num_data):

    xx[i] = data[i,0]
    yy[i] = data[i,1]
    zz[i] = data[i,2]
```

plot the data in the three dimensional space

In [3]:

```
fig = plt.figure(figsize=(12, 8))

ax1 = plt.subplot(111, projection='3d')
plt.title('data')

ax1.set_xlabel('$x$')
ax1.set_ylabel('$y$')
ax1.set_zlabel('$z$')

ax1.scatter(xx, yy, zz, marker='o', color='blue', alpha=0.2)

plt.tight_layout()
plt.show()
```



compute the loss function

In [4]:

```
def compute_residual(theta, x, y, z):
    num_data = x.shape[0]
    first = np.ones(num_data)
    X = np.column_stack([first, x, y])
    residual = z - X.dot(theta)
    return residual, num_data
```

In [5]:

```
def compute_loss(theta, x, y, z):
    residual, num_data = compute_residual(theta, x, y, z)

    loss = 0.0
    loss = np.sum(np.square(residual))/(2*num_data)
    return loss
```

compute the gradient for each model parameter (DO NOT COMPUTE THE GRADIENT FOR EACH MODEL PARAMETER, BUT DO COMPUTE THE GRADIENT OF THE MODEL PARAMTER VECTOR)

In [6]:

```
def compute_gradient(theta, x, y, z):
    one = np.ones(num_data)
    X = np.column_stack([one,x,y])
    f = np.dot(X.T,(X.dot(theta) - z))
    grad = f / num_data
    return grad
```

gradient descent for each model parameter

In [7]:

```
num_iteration   = 1000
learning_rate   = 0.01

theta           = np.array((0, 0, 0))
theta_iteration = np.zeros((num_iteration, theta.size))
loss_iteration  = np.zeros(num_iteration)

for i in range(num_iteration):
    theta = theta - learning_rate * compute_gradient(theta, xx, yy, zz)
    loss = compute_loss(theta, xx, yy, zz)
    theta_iteration[i] = theta
    loss_iteration[i] = loss
    print("iteration = %4d, loss = %5.5f" % (i, loss))
```

```
iteration =  150, loss = 7.33874
iteration =  151, loss = 7.33442
iteration =  152, loss = 7.33018
iteration =  153, loss = 7.32603
iteration =  154, loss = 7.32195
iteration =  155, loss = 7.31796
iteration =  156, loss = 7.31405
iteration =  157, loss = 7.31022
iteration =  158, loss = 7.30646
iteration =  159, loss = 7.30278
iteration =  160, loss = 7.29917
iteration =  161, loss = 7.29564
iteration =  162, loss = 7.29217
iteration =  163, loss = 7.28877
iteration =  164, loss = 7.28544
iteration =  165, loss = 7.28218
iteration =  166, loss = 7.27898
iteration =  167, loss = 7.27585
iteration =  168, loss = 7.27277
iteration =  169, loss = 7.26976
```

In [8]:

```
f = theta[0] + theta[1] * xx + theta[2] * yy
```

plot the results

In [9]:

```
def plot_loss_curve(loss_iteration):

    plt.figure(figsize=(8,6))
    plt.title('loss')
    # ===== FILL UP THE CODE =====
    plt.plot(loss_iteration, '-', color = 'red')
    # ============================
    plt.tight_layout()
    plt.show()
```

In [10]:

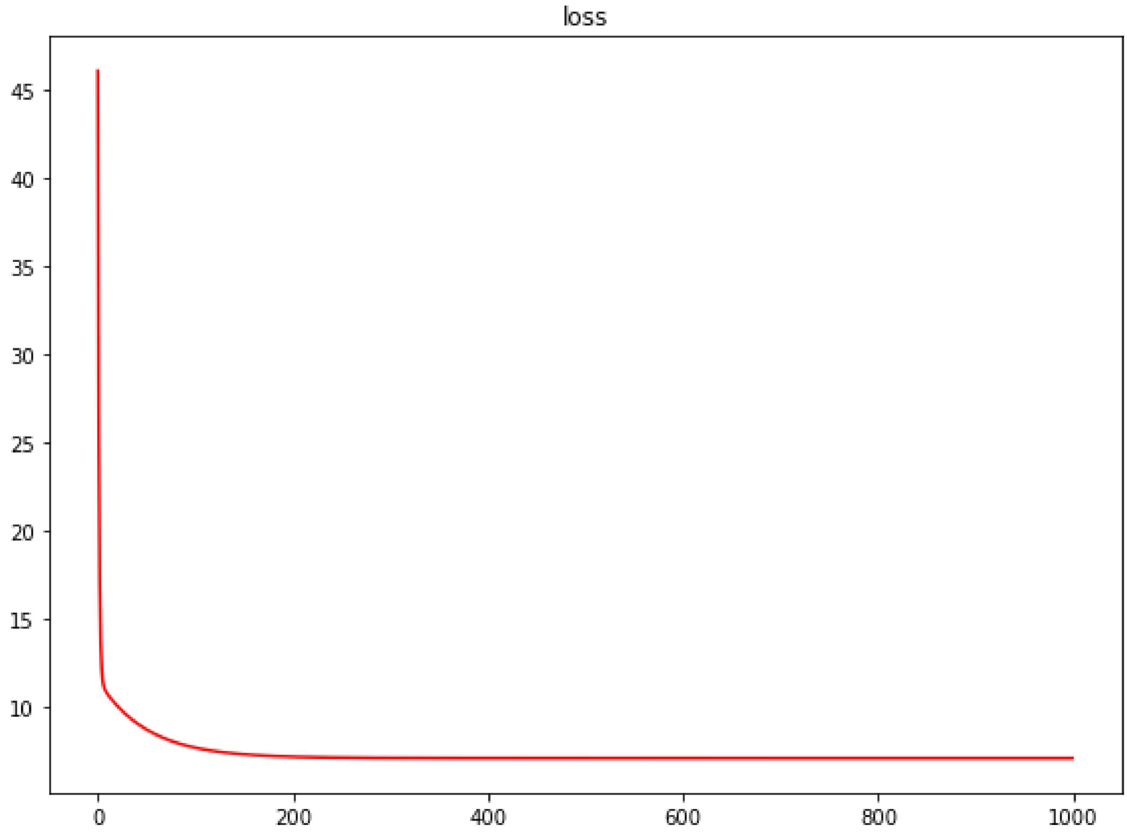
```
def plot_data(xx, yy, zz):

    fig = plt.figure(figsize=(12, 8))
    ax  = plt.subplot(111, projection='3d')
    plt.title('data')
    # ===== FILL UP THE CODE =====
    ax.scatter(xx,yy,zz,marker='o', color = 'blue', alpha = 0.2)
    # ============================
    plt.tight_layout()
    plt.show()
```

In [11]:

```

def plot_model_parameter(theta_iteration):

    plt.figure(figsize=(8,6))
    plt.title('model parameter')
    # ===== FILL UP THE CODE =====
    thetaT = theta_iteration.T
    plt.plot(thetaT[0], '-', color = 'blue')
    plt.plot(thetaT[1], '-', color = 'green')
    plt.plot(thetaT[2], '-', color = 'red')
    # ============================
    plt.tight_layout()
    plt.show()

```

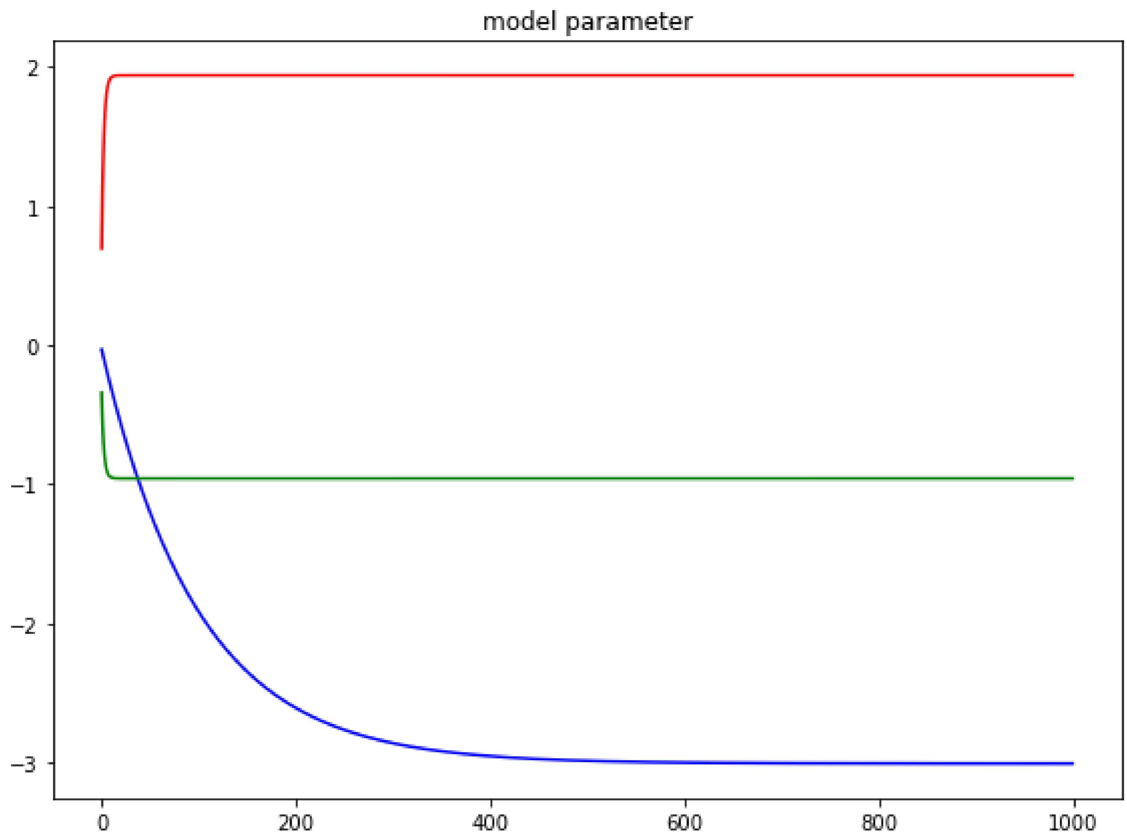
In [12]:

```
plot_loss_curve(loss_iteration)
```



In [13]:

```
plot_model_parameter(theta_iteration)
```

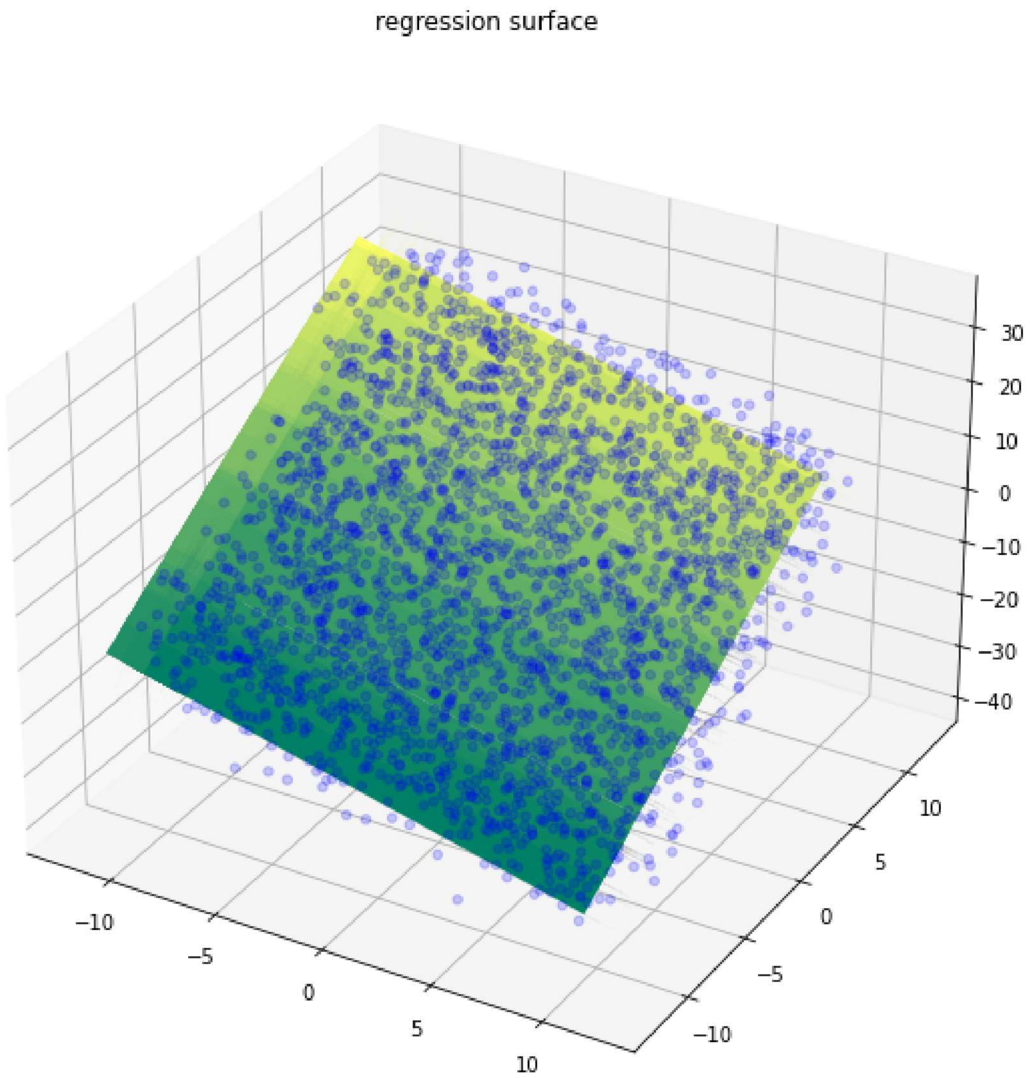


In [14]:

```
XX, YY = np.meshgrid(xx,yy)
ZZ = theta[0] + (theta[1]*XX) + (theta[2]*YY)

def plot_surface(XX, YY, ZZ, xx, yy, zz):
    fig = plt.figure(figsize=(8,8))
    ax  = fig.add_subplot(111, projection='3d')
    plt.title('regression surface')
    # ===== FILL UP THE CODE =====
    ax.scatter(xx, yy, zz, marker='o', color = 'blue', alpha = 0.2)
    surf = ax.plot_surface(XX, YY, ZZ, cmap='summer')
    # ============================
    plt.tight_layout()
    plt.show()
```

In [15]:

```
plot_surface(XX, YY, ZZ, xx, yy, zz)
```



* *

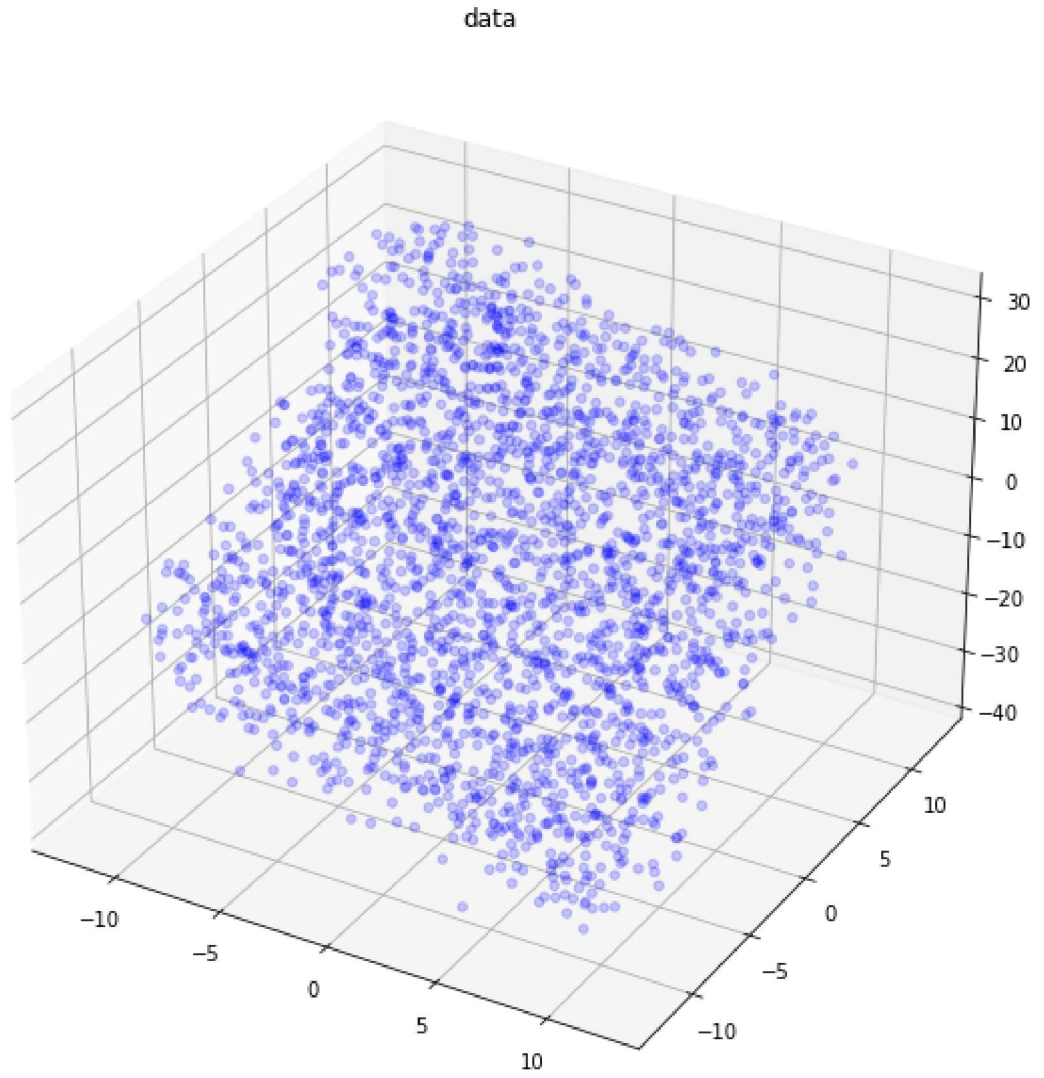
* results

*

01. plot the input data in blue point in 3-dimensional space

In [16]:

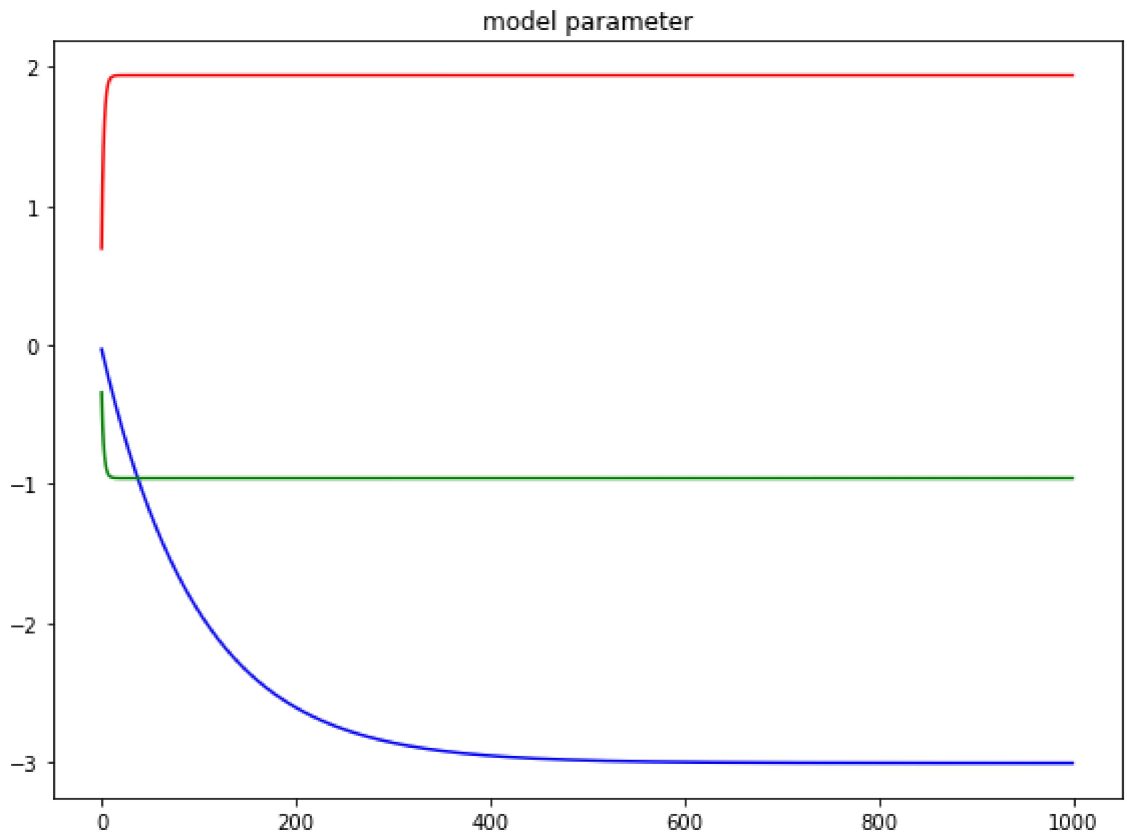
```
plot_data(xx, yy, zz)
```



02. plot the values of the model parameters θ_0 in red curve, θ_1 in green curve, and θ_2 in blue curve over the gradient descent iterations

In [17]:

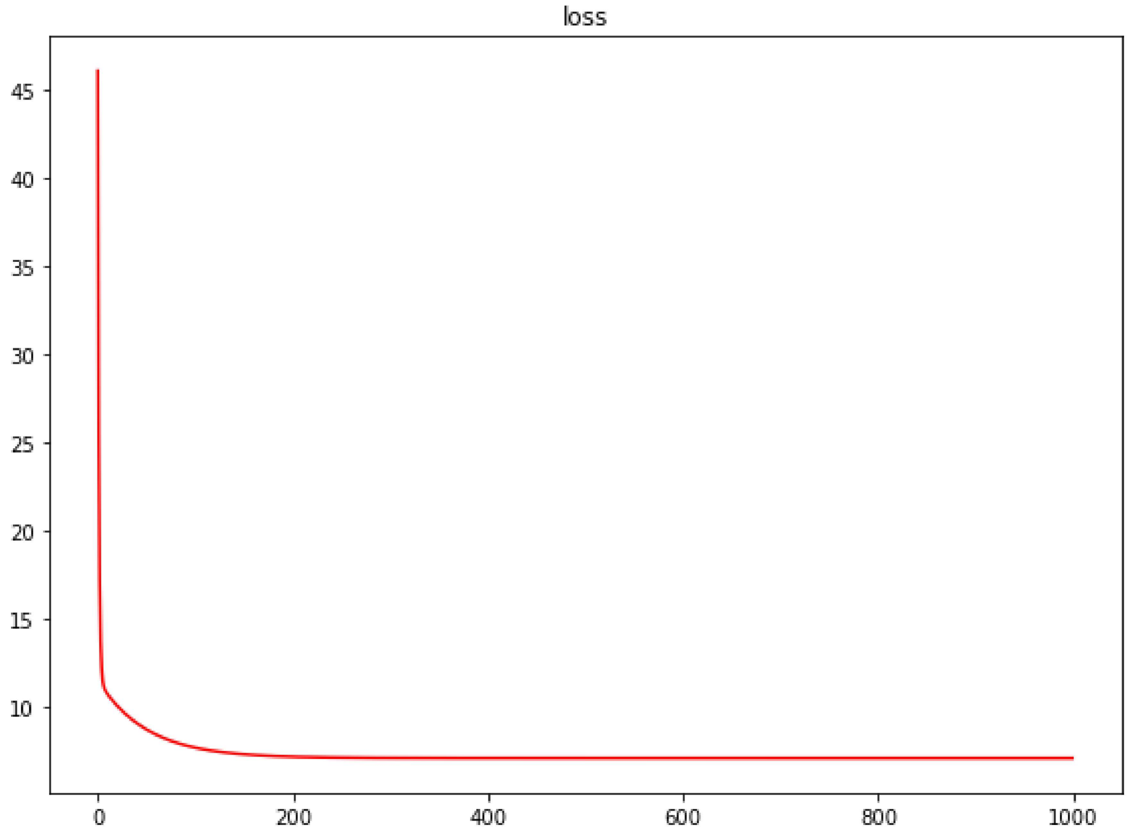
```
plot_model_parameter(theta_iteration)
```



03. plot the loss values $\mathcal{L}(\theta)$ in red curve over the gradient descent iterations

In [18]:

```
plot_loss_curve(loss_iteration)
```



04. plot the optimal regression surface $\hat{f}(\theta^*)$ in 3-dimensional space with a given set of data points superimposed

In [19]:

```
plot_surface(XX, YY, ZZ, xx, yy, zz)
```