In [123]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [124]:
```
df = pd.read_csv('D3.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
M
```

Out[124]: 100

In [125]:
```
X = df.values[:, 0]
K = df.values[:, 1]
Z = df.values[:, 2]
Y = df.values[:, 3]
m = len(Y)
X_0 = X
X_1 = K
X_2 = Z


m = len(Y) # Number of training examples
print('X = ', X[: 5]) # Show only first 5 records
print('Y = ', Y[: 5])
print('m = ', m)
```

```
X =  [0.         0.04040404 0.08080808 0.12121212 0.16161616]
Y =  [4.38754501 2.6796499  2.96848981 3.25406475 3.53637472]
m =  100
```

In [126]:
```
def gradient_descent(X, Y, theta, alpha, iterations):
    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
```

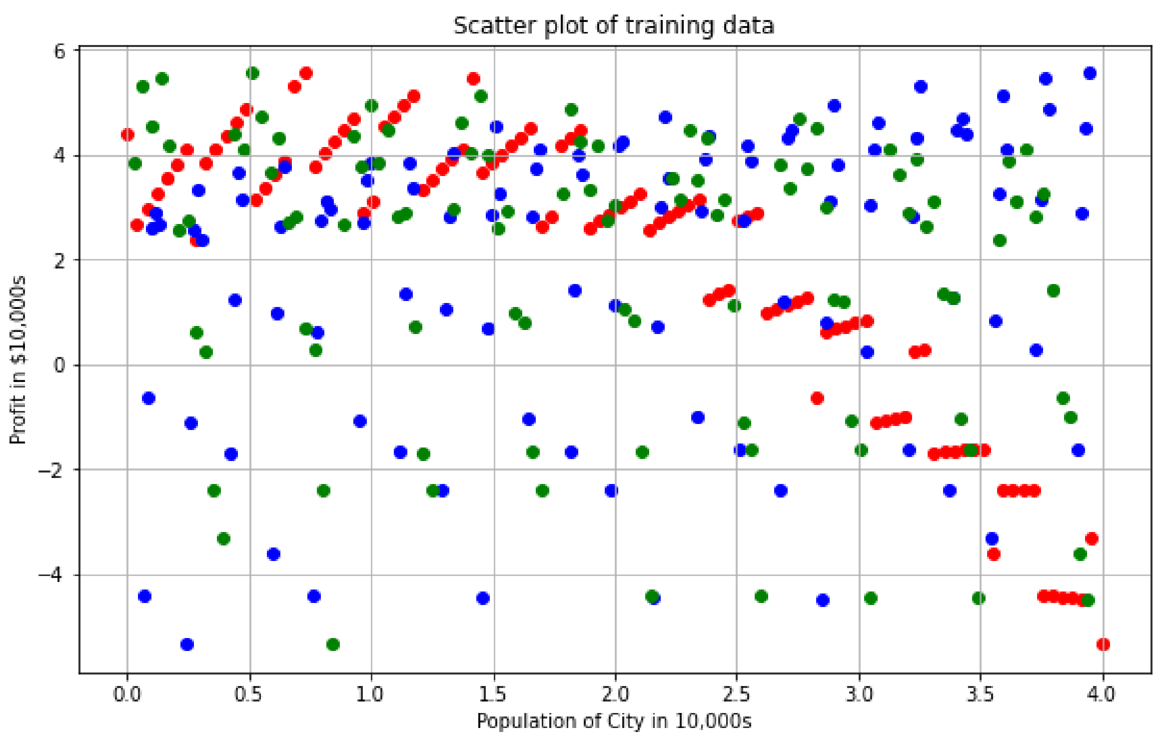
In [127]:
```
def compute_cost(X, Y, theta):
    predictions = X.dot(theta)
    errors = predictions - Y
    sqrErrors = np.square(errors)
    J = 1/(2*m)* np.sum(np.square(errors))
    return J
```

In [128]:

```
def DisplayData(X, color):
    plt.scatter(X,Y,color = color)
    plt.grid()
    plt.title('Scatter plot of training data')
    plt.rcParams["figure.figsize"] = (10,6)
    plt.xlabel('Population of City in 10,000s')
    plt.ylabel('Profit in $10,000s')
```

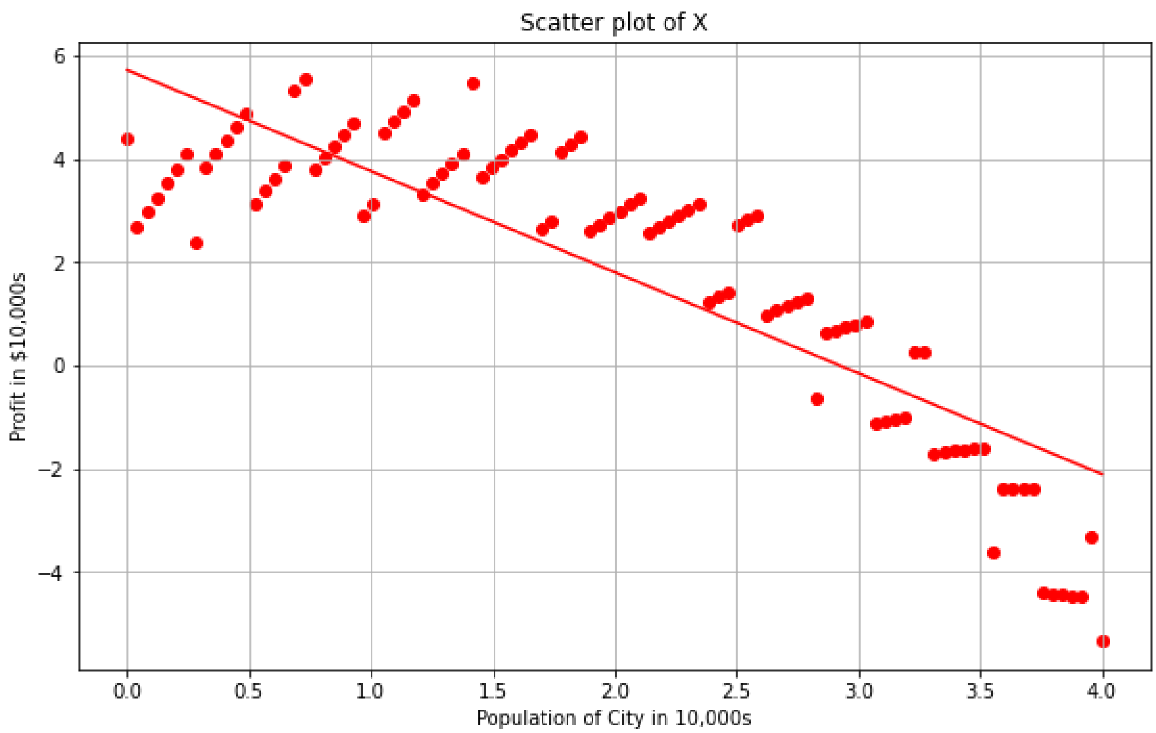
In [129]:

```
# Plot x
DisplayData(X, 'red')

# Plot k
DisplayData(K,'blue')

#Plot Z
DisplayData(Z,'green')
```



In [130]:
```
X0 = np.ones((m,1))
X1 = X_0.reshape(m,1)
X_1 = np.hstack((X0,X1))
theta = np.zeros(2)
iterations = 1500;
alpha = 0.01;
cost = compute_cost(X_1,Y,theta)
theta, cost_history = gradient_descent(X_1,Y,theta,alpha,iterations)

DisplayData(X, 'red')
plt.plot(X,X_1.dot(theta),color = 'red' ,label ='Linear Regression of K')
plt.title('Scatter plot of X')
```

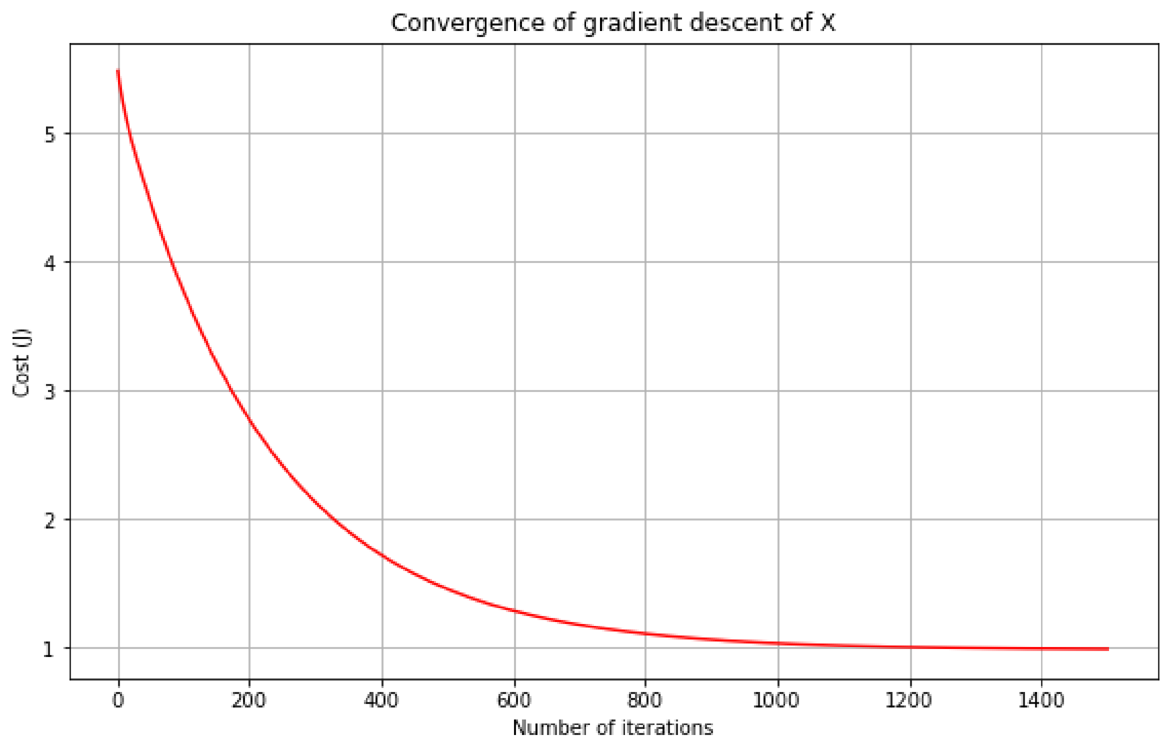
Out[130]: Text(0.5, 1.0, 'Scatter plot of X')



In [131]:
```
#Problem 1
```

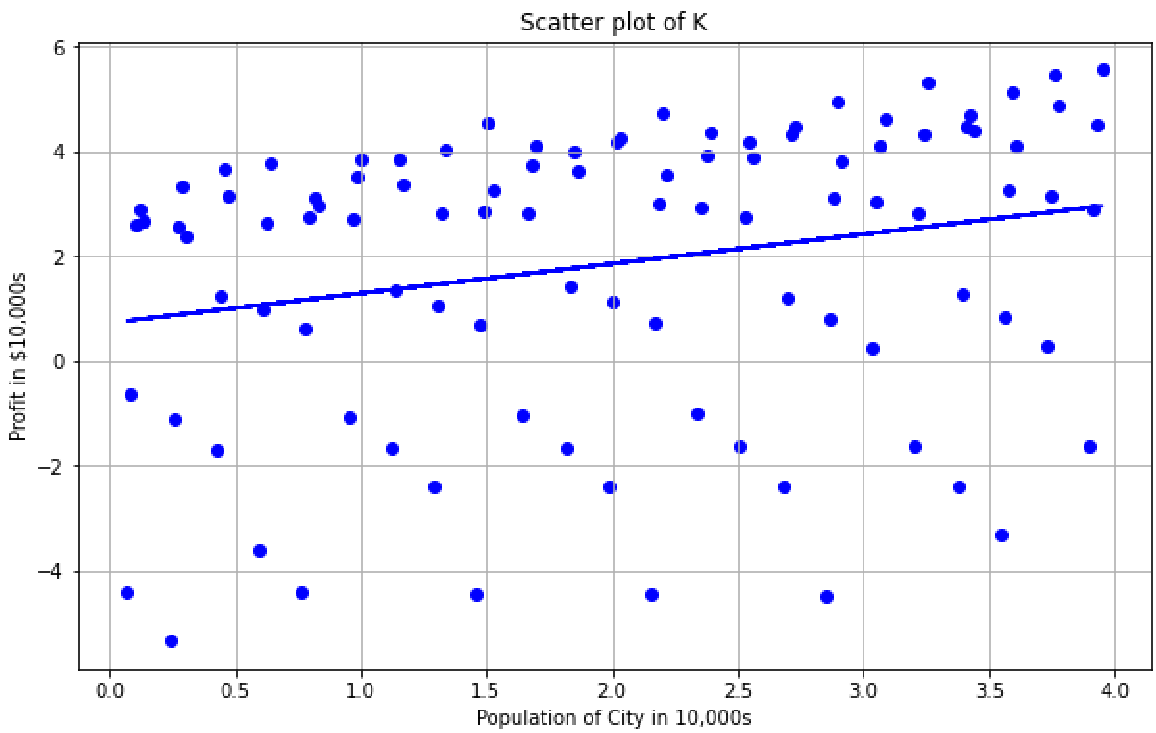
In [132]:
```
plt.plot(range(1, iterations + 1),cost_history, color='red')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent of X')
```

Out[132]:
```
Text(0.5, 1.0, 'Convergence of gradient descent of X')
```



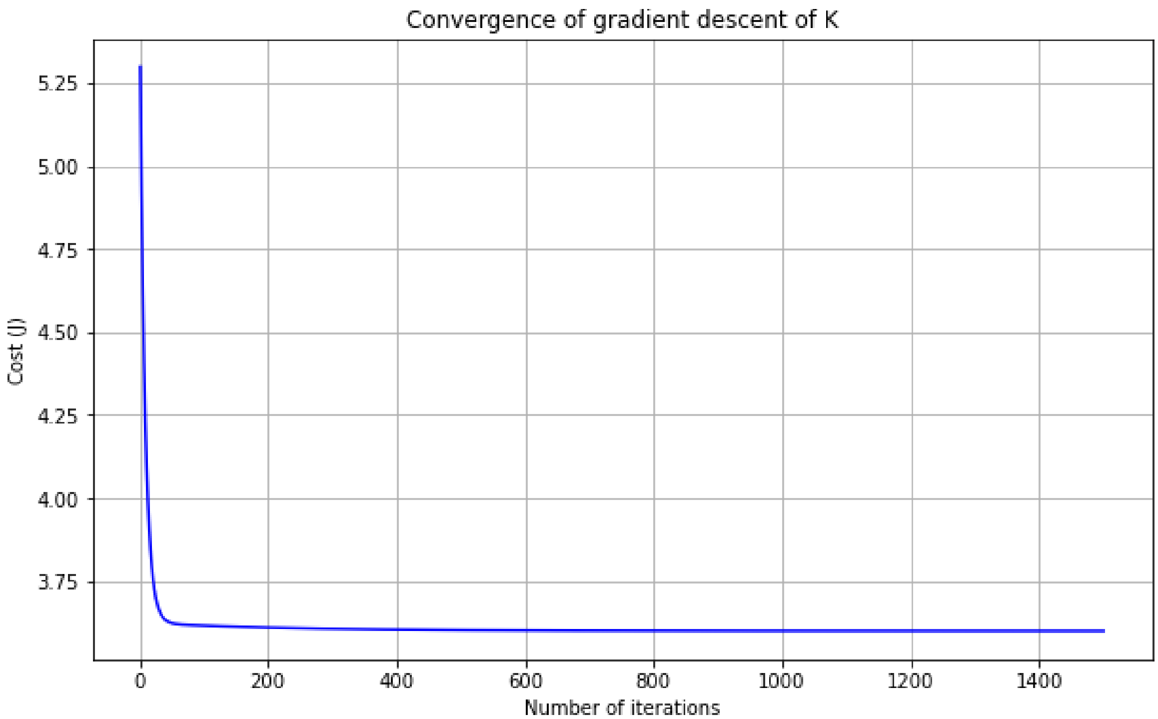
In [133]:
```
X0 = np.ones((m,1))
X1 = K.reshape(m,1)
X_1 = np.hstack((X0,X1))
theta = np.zeros(2)
iterations = 1500;
alpha = 0.01;
cost = compute_cost(X_1,Y,theta)
theta, cost_history = gradient_descent(X_1,Y,theta,alpha,iterations)

DisplayData(K, 'blue')
plt.plot(K,X_1.dot(theta),color = 'blue' ,label ='Linear Regression of K')
plt.title('Scatter plot of K')
```

Out[133]: Text(0.5, 1.0, 'Scatter plot of K')



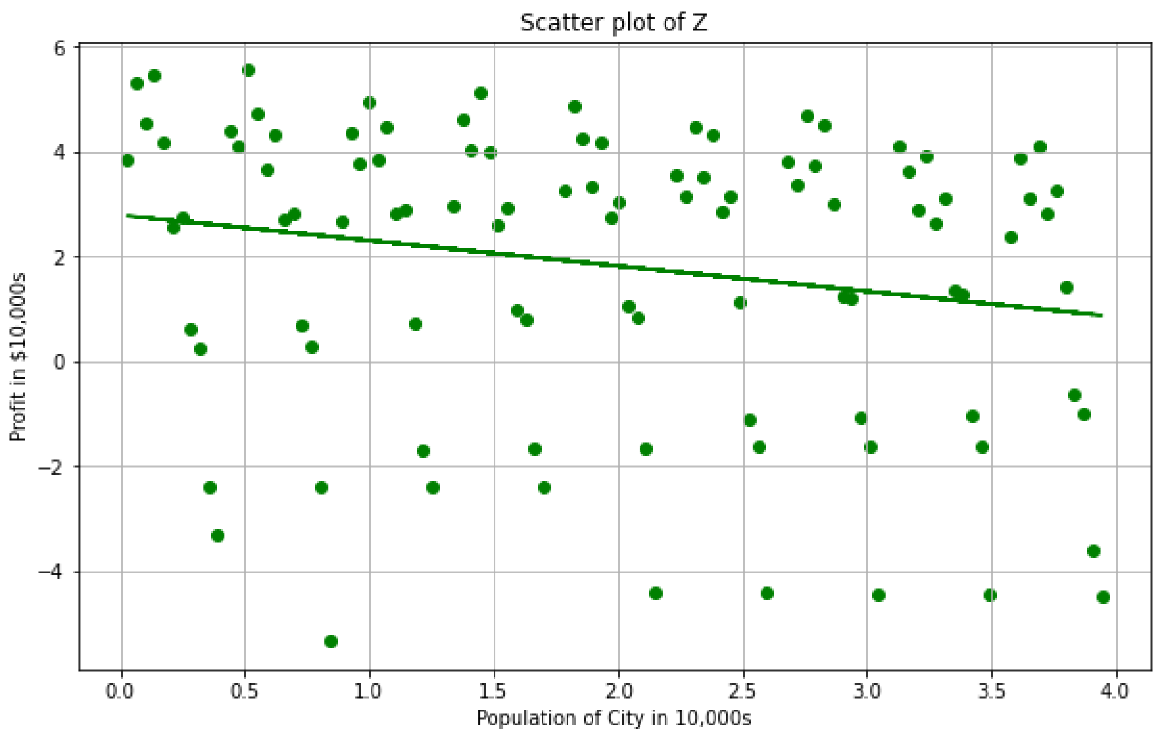
In [134]:
```
plt.plot(range(1,iterations+1),cost_history,color = 'blue')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent of K')
```

Out[134]:
```
Text(0.5, 1.0, 'Convergence of gradient descent of K')
```



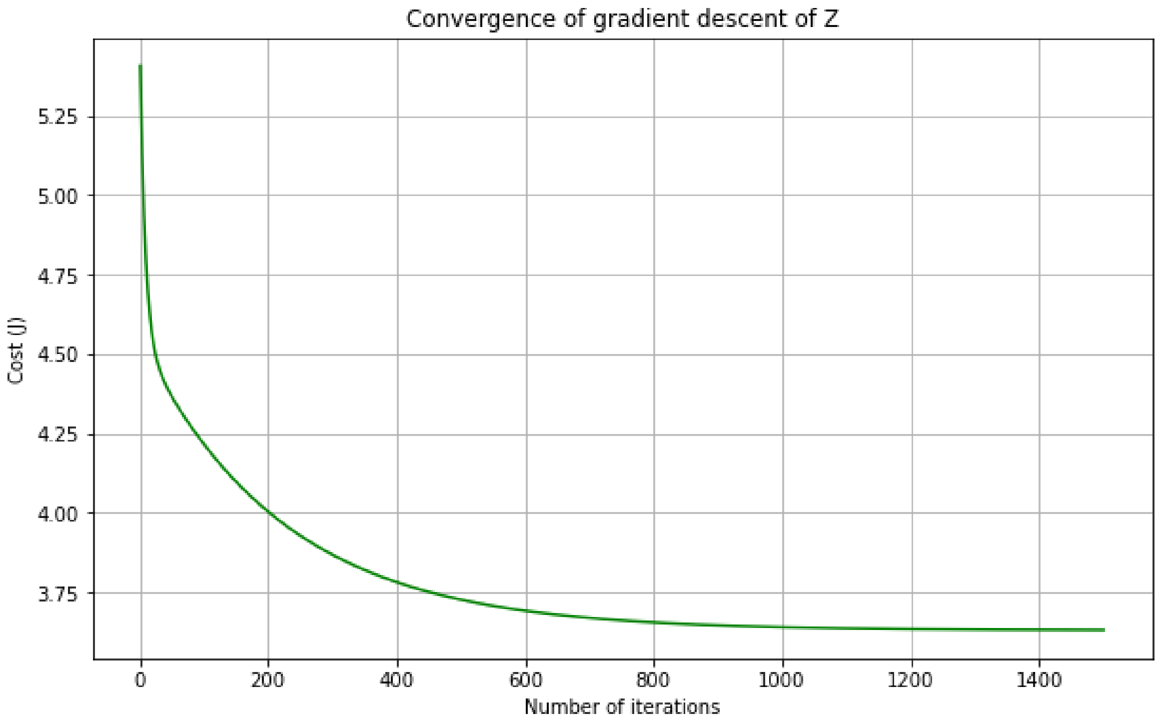
In [135]:
```
X0 = np.ones((m,1))
X2 = Z.reshape(m,1)
X_2 = np.hstack((X0,X2))
theta = np.zeros(2)
iterations = 1500;
alpha = 0.01;

cost = compute_cost(X_2,Y,theta)
theta, cost_history = gradient_descent(X_2,Y,theta,alpha,iterations)

DisplayData(Z, 'GREEN')
plt.plot(Z,X_2.dot(theta),color = 'green' ,label ='Linear Regression of Z')
plt.title('Scatter plot of Z')
```

Out[135]: Text(0.5, 1.0, 'Scatter plot of Z')



In [136]:
```
plt.plot(range(1,iterations+1),cost_history,color = 'green')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent of Z')
```

Out[136]:
```
Text(0.5, 1.0, 'Convergence of gradient descent of Z')
```



In [137]:

```
'''
Problem 1
Problem 1 Q3
#Which explanatory variable has the lower loss (cost) for explaining the outp
Variable K or X_1 had the steepest slope, which means it has the lowest cost

Problem 1 Q4
#Based on your training observations, describe the impact of the different le
A higher learning late has a steeper curve which allows for less iterations
'''
```

Out[137]:

```
'\nProblem 1\nProblem 1 Q3\n#Which explanatory variable has the lower loss (cost) for explaining the output (Y)?\nVariable K or X_1 had the steepest slope, which means it has the lowest cost\n\nProblem 1 Q4\n#Based on your training observations, describe the impact of the different learning rates on the final loss and number of training iteration.\nA higher learning late has a steeper curve which allows for less iterations \n'
```

In [138]:

```
'''
Problem 2
Problem 2 Q3
#Based on your training observations, describe the impact of the different le
If the learning late is to low than the line will stabalize earlier. A learni
'''
```

Out[138]:

```
'\nProblem 2\nProblem 2 Q3\n#Based on your training observations, describe the impact of the different learning rates on the final loss and number of training iteration.\nIf the learning late is to low than the line will stabalize earlier. A learning late needs to be higher if possible.\n'
```

In [139]:

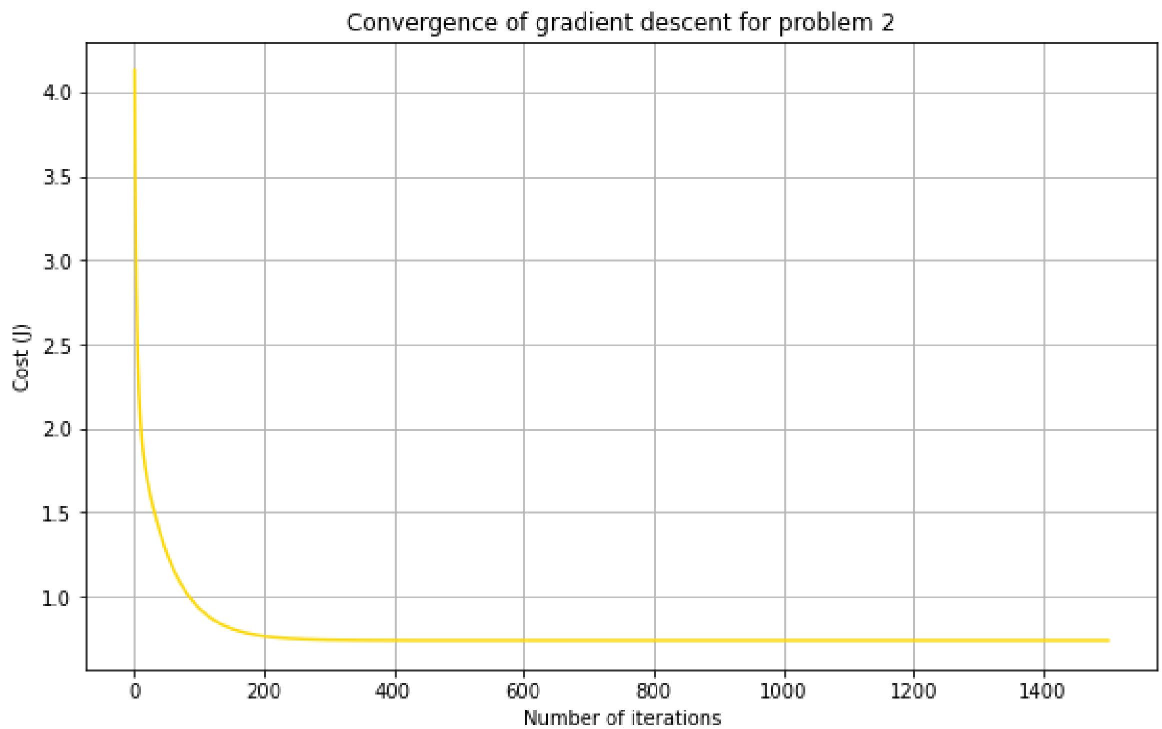
```
def test():
    X0 = np.ones((m,1))
    X1 = X.reshape(m, 1)
    X2 = K.reshape(m,1)
    X3 = Z.reshape(m,1)
    X_4 = np.hstack((X0, X1, X2, X3))
    theta = np.zeros(4)
    iterations = 1500;
    alpha = 0.1;

    cost = compute_cost(X_4,Y,theta)
    theta, cost_history = gradient_descent(X_4,Y,theta,alpha,iterations)
    return theta, cost_history
```

In [140]:
```
theta, cost_history = test()
theta
```

Out[140]: array([5.31416563, -2.00371905, 0.53256359, -0.26560164])

In [141]:
```
plt.plot(range(1,iterations+1),cost_history,color = 'gold')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent for problem 2')
```

Out[141]: Text(0.5, 1.0, 'Convergence of gradient descent for problem 2')



In [142]:
```
X1 = theta[0] + (1)*theta[1] + (1)*theta[2] + (1)*theta[3]
X2 = theta[0] + (2)*theta[1] + (0)*theta[2] + (4)*theta[3]
X3 = theta[0] + (3)*theta[1] + (2)*theta[2] + (1)*theta[3]
print('X1 = ', X1) # Show only first 5 records
print('X2 = ', X2)
print('X3 = ', X3)
```

```
X1 =  3.577408529345461
X2 =  0.2443209702176521
X3 =  0.10253401973591902
```

In [ ]: