```
import numpy as np
import matplotlib.pyplot as plt

def mean_squared_error(y_true, y_predicted):
  cost = np.sum((y_true-y_predicted)**2) / len(y_true)
  return cost


def gradient_descent(x, y, iterations = 1000, learning_rate = 0.009, stopping_threshold = 1e-6):
  current_weight = 0.1
  current_bias = 0.01
  iterations = iterations
  learning_rate = learning_rate
  n = float(len(x))

  costs = []
  weights = []
  previous_cost = None


  for i in range(iterations):

    y_predicted = (current_weight * x) + current_bias

    current_cost = mean_squared_error(y, y_predicted)
    if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
      break

    previous_cost = current_cost

    costs.append(current_cost)
    weights.append(current_weight)


    weight_derivative = -(2/n) * sum(x * (y-y_predicted))
    bias_derivative = -(2/n) * sum(y-y_predicted)


    current_weight = current_weight - (learning_rate * weight_derivative)
    current_bias = current_bias - (learning_rate * bias_derivative)


    print("Iteration",i+1, "Cost: ",current_cost, "Weight: ",current_weight,"Bias: ",current_bias)

  return current_weight, current_bias



X = np.array([1,3.5,6])
Y = np.array([4,5.5,9])

estimated_weight, estimated_bias = gradient_descent(X, Y, iterations=1000)
print(f"\nEstimated Weight: {estimated_weight}\nEstimated Bias: {estimated_bias}")


Y_pred = estimated_weight*X + estimated_bias

plt.figure(figsize = (8,6))
plt.scatter(X, Y, marker='o', color='red')
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='blue',markerfacecolor='red', markersize=10,)
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

```
Iteration 975 Cost:  0.2225007978226865 Weight:  1.0071355015027494 Bias:  2.63369
Iteration 976 Cost:  0.22249837002655445 Weight:  1.007104340388653 Bias:  2.63383
Iteration 977 Cost:  0.2224959633887503 Weight:  1.0070733153567972 Bias:  2.63397
Iteration 978 Cost:  0.2224935777248799 Weight:  1.0070424258129034 Bias:  2.63412
Iteration 979 Cost:  0.22249121285215367 Weight:  1.0070116711652888 Bias:  2.6342
Iteration 980 Cost:  0.22248886858937586 Weight:  1.0069810508248538 Bias:  2.6344
Iteration 981 Cost:  0.22248654475693083 Weight:  1.006950564205072 Bias:  2.63454
Iteration 982 Cost:  0.22248424117676635 Weight:  1.0069202107219783 Bias:  2.6346
Iteration 983 Cost:  0.22248195767238468 Weight:  1.0068899897941577 Bias:  2.6348
Iteration 984 Cost:  0.2224796940688235 Weight:  1.0068599008427341 Bias:  2.63496
Iteration 985 Cost:  0.2224774501926464 Weight:  1.0068299432913597 Bias:  2.63510
Iteration 986 Cost:  0.22247522587192936 Weight:  1.0068001165662035 Bias:  2.6352
Iteration 987 Cost:  0.2224730209362449 Weight:  1.0067704200959404 Bias:  2.63537
Iteration 988 Cost:  0.22247083521665287 Weight:  1.0067408533117406 Bias:  2.6355
Iteration 989 Cost:  0.2224686685456836 Weight:  1.0067114156472576 Bias:  2.63565
Iteration 990 Cost:  0.2224665207573281 Weight:  1.0066821065386193 Bias:  2.63578
Iteration 991 Cost:  0.22246439168702314 Weight:  1.0066529254244154 Bias:  2.6359
Iteration 992 Cost:  0.22246228117164066 Weight:  1.0066238717456872 Bias:  2.6360
Iteration 993 Cost:  0.22246018904947376 Weight:  1.0065949449459173 Bias:  2.6361
Iteration 994 Cost:  0.22245811516022532 Weight:  1.0065661444710186 Bias:  2.6363
Iteration 995 Cost:  0.22245605934499366 Weight:  1.0065374697693237 Bias:  2.6364
Iteration 996 Cost:  0.2224540214462644 Weight:  1.0065089202915742 Bias:  2.63658
Iteration 997 Cost:  0.22245200130789397 Weight:  1.0064804954909106 Bias:  2.6367
Iteration 998 Cost:  0.2224499987751001 Weight:  1.0064521948228615 Bias:  2.63685
Iteration 999 Cost:  0.22244801369445022 Weight:  1.006424017745333 Bias:  2.63698
Iteration 1000 Cost:  0.22244604591384787 Weight:  1.0063959637185989 Bias:  2.637

Estimated Weight: 1.0063959637185989
Estimated Bias: 2.637109908483543
```