Abdullah Abdelhakeem Amer (HW4)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.metrics import r2_score
import random
random.seed(0)
```

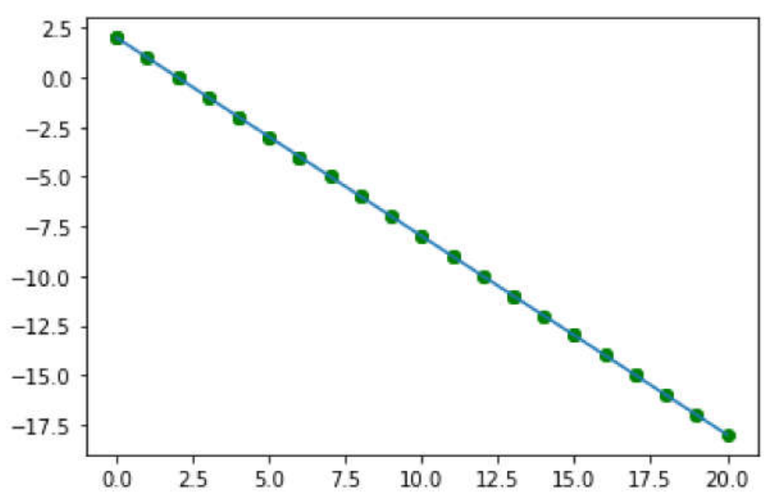
In [21]:
```
x=np.linspace(0,20,dtype=int)
#x=[random.randrange(0,20,1) for i in range(20)]
x=np.array(x)
y = -1*x + 2

print(f'x={x}\ny={y}')
print(f'x_shape={x.shape}\ny_shape={y.shape}')
```

```
x=[ 0  0  0  1  1  2  2  2  3  3  4  4  4  5  5  6  6  6  7  7  8  8  8  9
  9 10 10 11 11 11 12 12 13 13 13 14 14 15 15 15 16 16 17 17 17 18 18 19
 19 20]
y=[  2   2   2   1   1   0   0   0  -1  -1  -2  -2  -2  -3  -3  -4  -4  -4
  -5  -5  -6  -6  -6  -7  -7  -8  -8  -9  -9  -9 -10 -10 -11 -11 -11 -12
 -12 -13 -13 -13 -14 -14 -15 -15 -15 -16 -16 -17 -17 -18]
x_shape=(50,)
y_shape=(50,)
```

In [ ]:

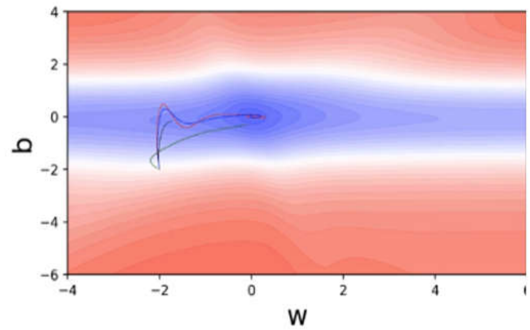
Plot your data points.

In [22]:
```
plt.scatter(x,y,color="green")
plt.plot(x,y)
plt.show()
```



Motivation

- For the real-time datasets, most of the features are sparse **i.e. having zero values**.
- Due to this for most of the cases, the corresponding gradient is zero and therefore the parameters update is also zero.
- To resonate this problem, these update should be boosted i.e. a high learning rate for sparse features.
- The learning rate should be adaptive for fairly sparse data.



If we are dealing with sparse features then learning rate should be high whereas for dense features learning rate should be low.

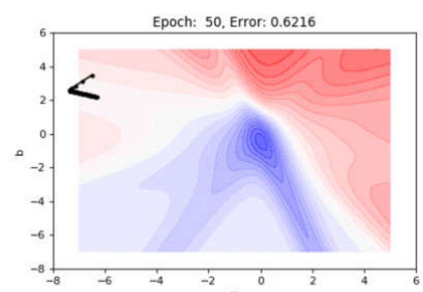
6

Adagrad

- **Adagrad** adopts the learning rate**(η)** based on the sparsity of features. So, the parameters with small updates **(sparse features)** have high learning rate whereas the parameters with large updates **(dense features)** have low learning rate.
- **v(t)** accumulates the running sum of square of the gradients. Square of **∇w(t)** neglects the sign of gradients.
- **v(t)** indicates accumulated gradient up to time **t**.
- **Epsilon (ε)** in the denominator avoids the chances of divide by zero error.
- if **v(t)** is low (due to less update up to time **t**) for a parameter then the effective learning rate will be high and if **v(t)** is high for a parameter then effective learning rate will be less.

$$v_t = v_{t-1} + (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t} + \varepsilon} \nabla w_t$$



9

In [23]:
```
def ADAGRAD(x,y,maxEpochs,learningRate ,convergence , epsilon):
    loss=[]
    thetaList0=[]
    thetaList1=[]
    ypredictedEpochs=[]
    X=np.column_stack((np.ones(len(x),dtype=int),x)) #more columns x0 ,x1
    y=y.reshape(-1,1)     #(shape(20,1))
    m=(X.shape)[0]        #m=20
    thetas=np.zeros((X.shape[1],1))
    count=0
    epoch=0

    v=0
    while epoch < maxEpochs:
        count +=1

        ypredicted = X @ thetas # (20,2) @ (2,1) ===> (20,1)
        costOld=(np.sum(np.square(ypredicted - y)))/ (2*m) #Mean Square Error (ol

        Gradient = (np.transpose(X) @ (ypredicted - y) ) / m # (2,20) @ (20,1) ==

        v= v + np.square(Gradient)


        thetas =thetas - ((learningRate * Gradient) / (np.sqrt(v) + epsilon)) #(2
        thetaList0.append(thetas[0])
        thetaList1.append(thetas[1])


        ypredicted = X @ thetas # (20,2) @ (2,1) ===> (20,1)
        costNew=(np.sum(np.square(ypredicted - y)))/ (2*m) #Mean Square Error (Ne

        loss.append(costNew) #loss list
        ypredictedEpochs.append(ypredicted)

        #print(np.linalg.norm(Gradient))
        if abs(costOld - costNew) < convergence:
            print(f'convergence occur after ({count}) iterations')
            return r2_score(y,ypredicted) ,thetas ,ypredicted ,loss ,thetaList0 ,

        epoch+=1

    print(f'sorry Max_epochs ({maxEpochs}) have occured')
    return r2_score(y,ypredicted),thetas ,ypredicted ,loss ,thetaList0 ,thetaList
```

In [24]:
```
R2Score,thetas,ypredicted,loss,thetaList0,thetaList1,ypredictedEpochs=ADAGRAD(x,y
```

```
convergence occur after (114) iterations
```

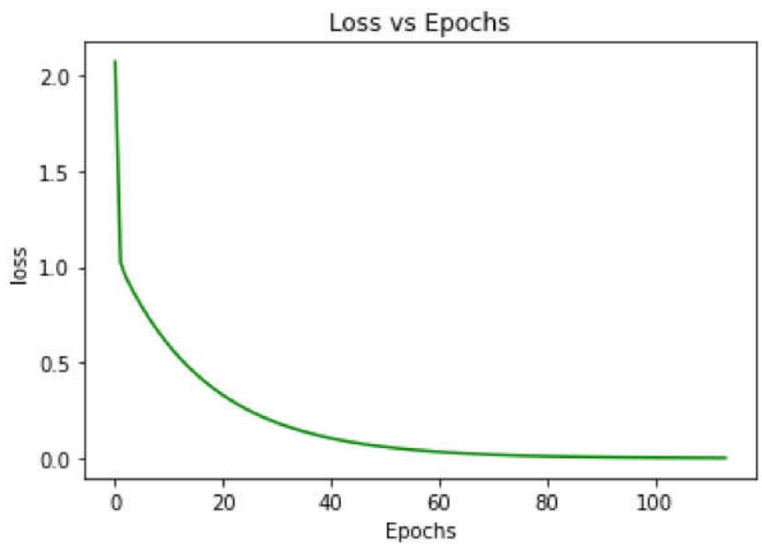
In [25]:
```
R2Score
```

Out[25]:
```
0.9999036347772027
```

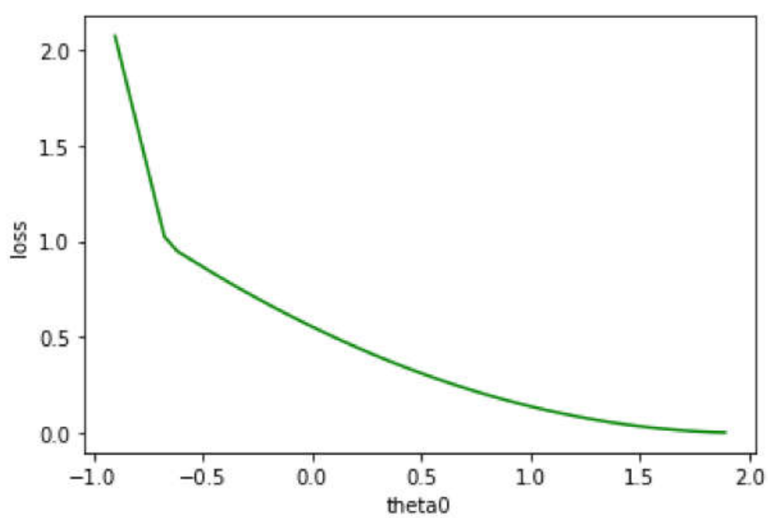
In [26]:
```
thetas
```

Out[26]:
```
array([[ 1.8902315 ],
       [-0.99143089]])
```

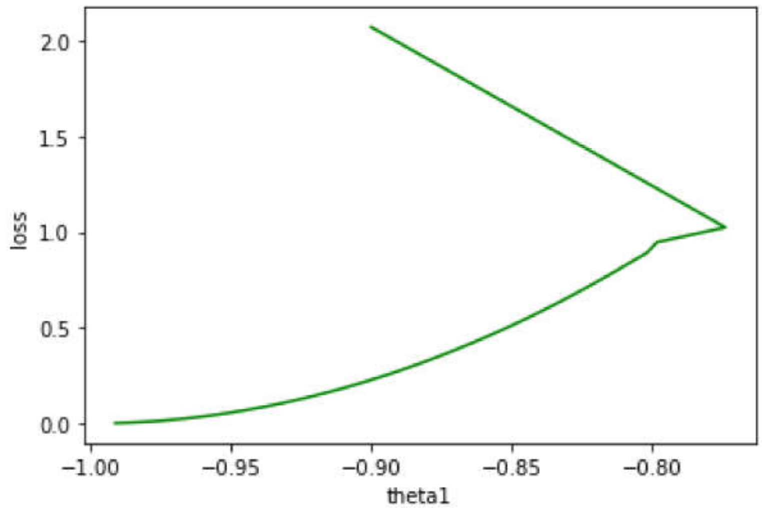
In [27]:
```
plt.plot(loss , color="green")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.title("Loss vs Epochs")
plt.show()
```



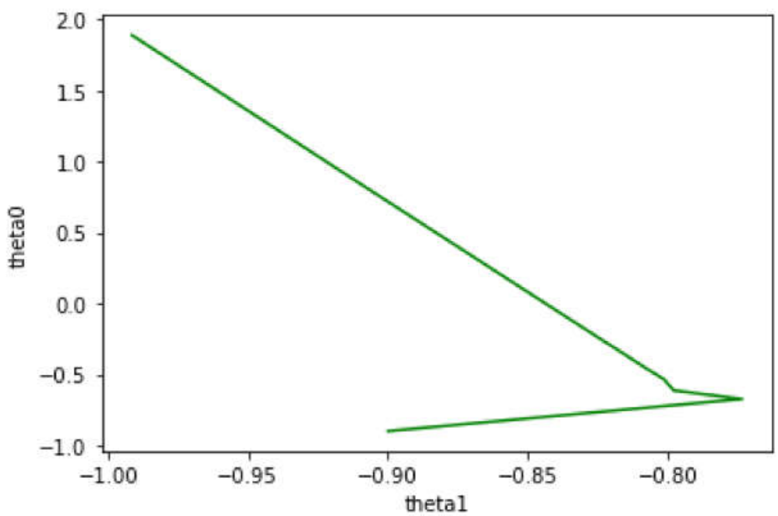
In [28]:
```
plt.plot(thetaList0,loss,color="green")
plt.xlabel("theta0")
plt.ylabel("loss")
plt.show()
```



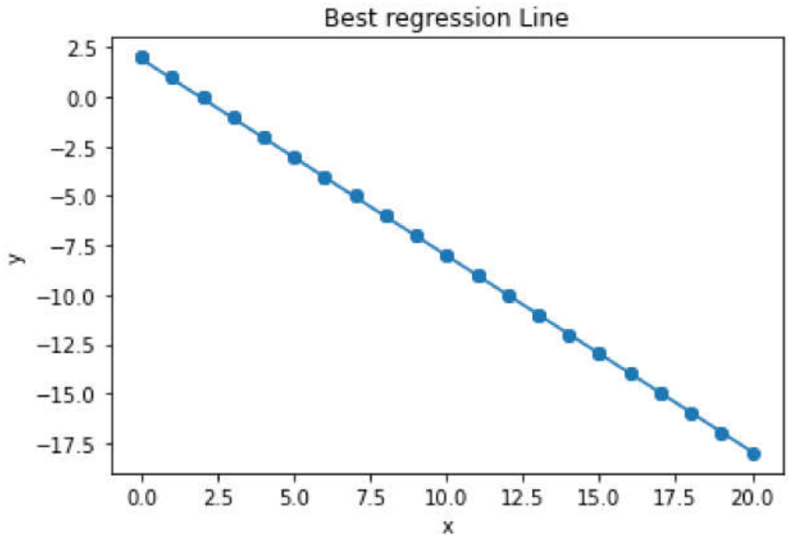
In [29]:
```
plt.plot(thetaList1,loss,color="green")
plt.xlabel("theta1")
plt.ylabel("loss")
plt.show()
```



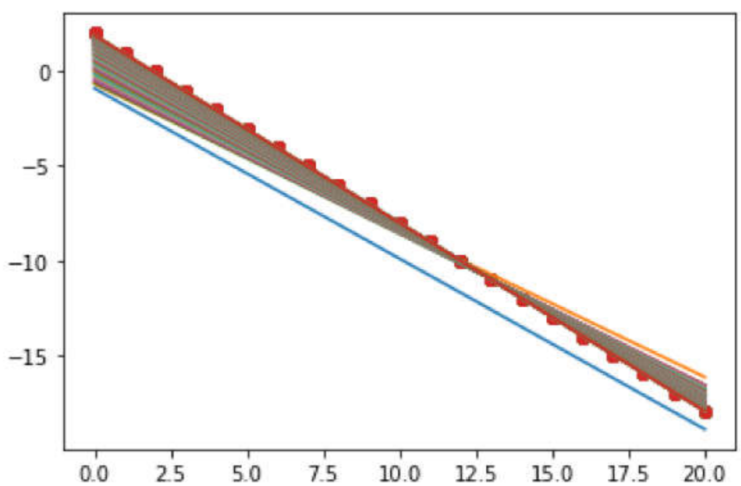
In [30]:
```
plt.plot(thetaList1,thetaList0,color="green")
plt.xlabel("theta1")
plt.ylabel("theta0")
plt.show()
```



In [31]:
```
plt.scatter(x,y)
plt.plot(x,ypredicted)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Best regression Line")
plt.show()
```



In [32]:
```
for h in ypredictedEpochs:
    plt.scatter(x,y)
    plt.plot(x,h)
```



In [ ]:

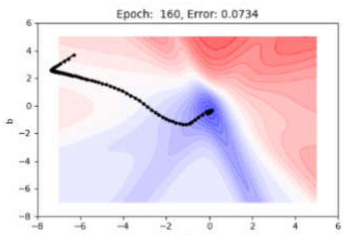
In [ ]:

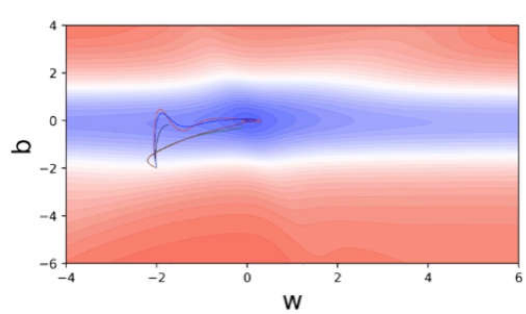
RMSProp

$$v_t = \beta * v_{t-1} + (1-\beta)(\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \varepsilon}} \nabla w_t$$

- **RMSProp** Overcomes the decaying learning rate problem of **adagrad** and prevents the rapid growth in **v(t)**.
- Instead of accumulating squared gradients from the beginning, it accumulates the previous gradients in some portion(weight).
- **v(t)** is exponentially decaying average of all the previous squared gradients.
- Prevents rapid growth of **v(t)**.
- The algorithm keeps learning and tries to converge.





13

In [ ]:

In [33]:
```
def RMSPROP(x,y,maxEpochs, beta , learningRate ,convergence , epsilon):
    loss=[]
    thetaList0=[]
    thetaList1=[]
    ypredictedEpochs=[]
    X=np.column_stack((np.ones(len(x),dtype=int),x)) #more columns x0 ,x1
    y=y.reshape(-1,1)     #(shape(20,1))
    m=(X.shape)[0]        #m=20
    thetas=np.zeros((X.shape[1],1))
    count=0
    epoch=0

    v=0
    while epoch < maxEpochs:
        count +=1

        ypredicted = X @ thetas # (20,2) @ (2,1) ===> (20,1)
        costOld=(np.sum(np.square(ypredicted - y)))/ (2*m) #Mean Square Error (ol

        Gradient = (np.transpose(X) @ (ypredicted - y) ) / m # (2,20) @ (20,1) ==

        v= (beta * v) +  ((1-beta)* (np.square(Gradient)))


        thetas =thetas - ((learningRate * Gradient) / (np.sqrt(v) + epsilon)) #(2
        thetaList0.append(thetas[0])
        thetaList1.append(thetas[1])


        ypredicted = X @ thetas # (20,2) @ (2,1) ===> (20,1)
        costNew=(np.sum(np.square(ypredicted - y)))/ (2*m) #Mean Square Error (Ne

        loss.append(costNew) #loss list
        ypredictedEpochs.append(ypredicted)

        if abs(costOld - costNew) < convergence:
            print(f'convergence occur after ({count}) iterations')
            return r2_score(y,ypredicted) ,thetas ,ypredicted ,loss ,thetaList0 ,

        epoch+=1

    print(f'sorru=y Max_epochs ({maxEpochs}) have occured')
    return r2_score(y,ypredicted),thetas ,ypredicted ,loss ,thetaList0 ,thetaList
```

In [34]:
```
R2Score,thetas,ypredicted,loss,thetaList0,thetaList1,ypredictedEpochs=RMSPROP(x,y
```

```
convergence occur after (322) iterations
```

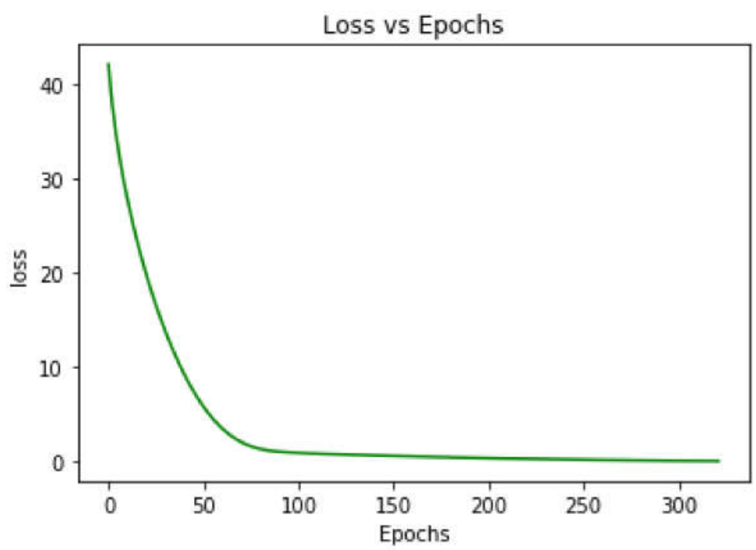
In [35]:
```
R2Score
```

Out[35]:
```
0.9982540515900505
```

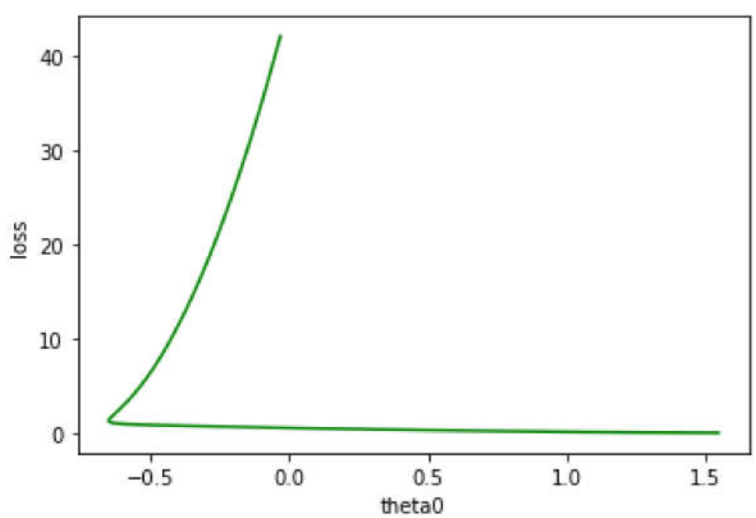
In [36]:
```
thetas
```

Out[36]:
```
array([[ 1.54695258],
       [-0.96011771]])
```

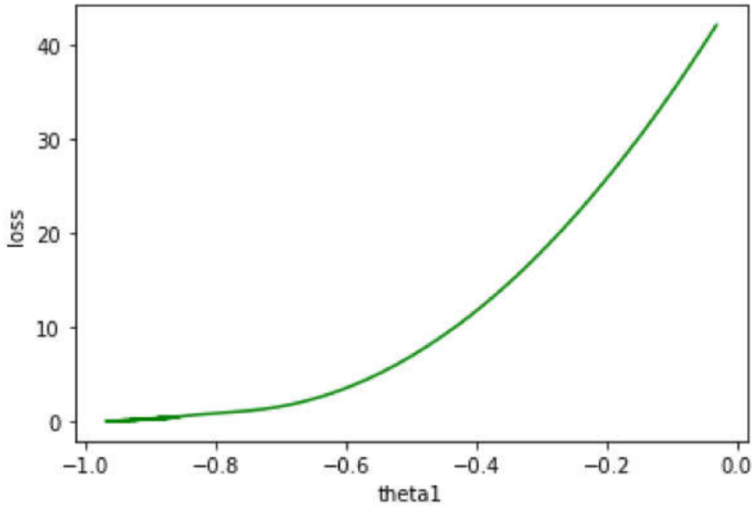
In [37]:
```
plt.plot(loss , color="green")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.title("Loss vs Epochs")
plt.show()
```



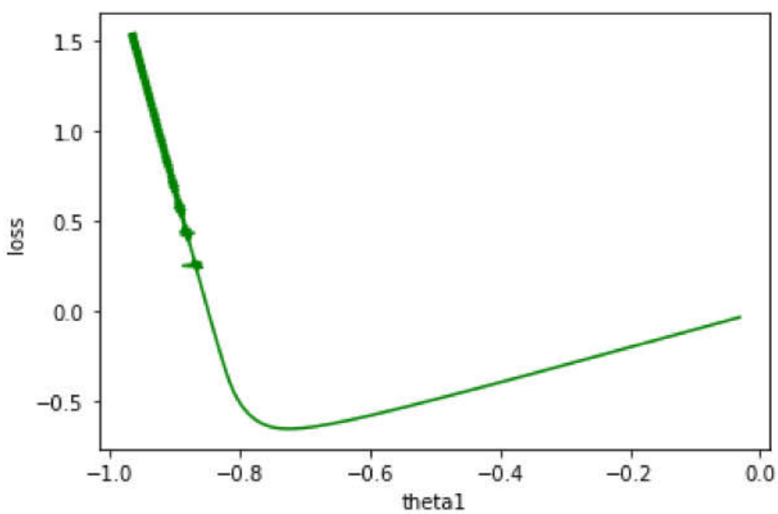
In [38]:
```
plt.plot(thetaList0,loss,color="green")
plt.xlabel("theta0")
plt.ylabel("loss")
plt.show()
```



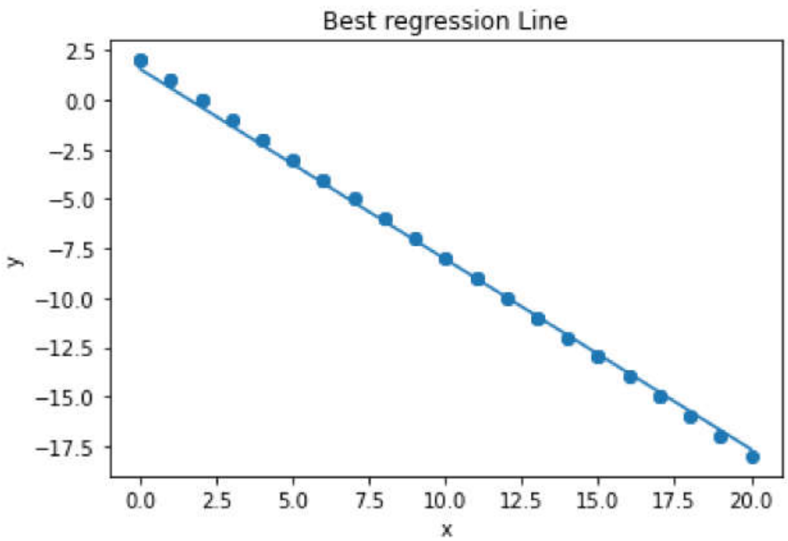
In [39]:
```
plt.plot(thetaList1,loss,color="green")
plt.xlabel("theta1")
plt.ylabel("loss")
plt.show()
```



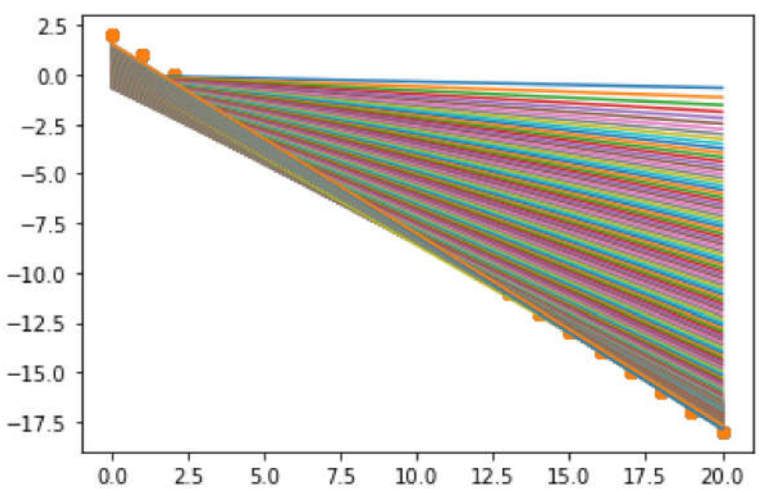
In [40]:
```
plt.plot(thetaList1,thetaList0,color="green")
plt.xlabel("theta1")
plt.ylabel("loss")
plt.show()
```



In [41]:
```
plt.scatter(x,y)
plt.plot(x,ypredicted)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Best regression Line")
plt.show()
```



In [42]:
```
for h in ypredictedEpochs:
    plt.scatter(x,y)
    plt.plot(x,h)
```



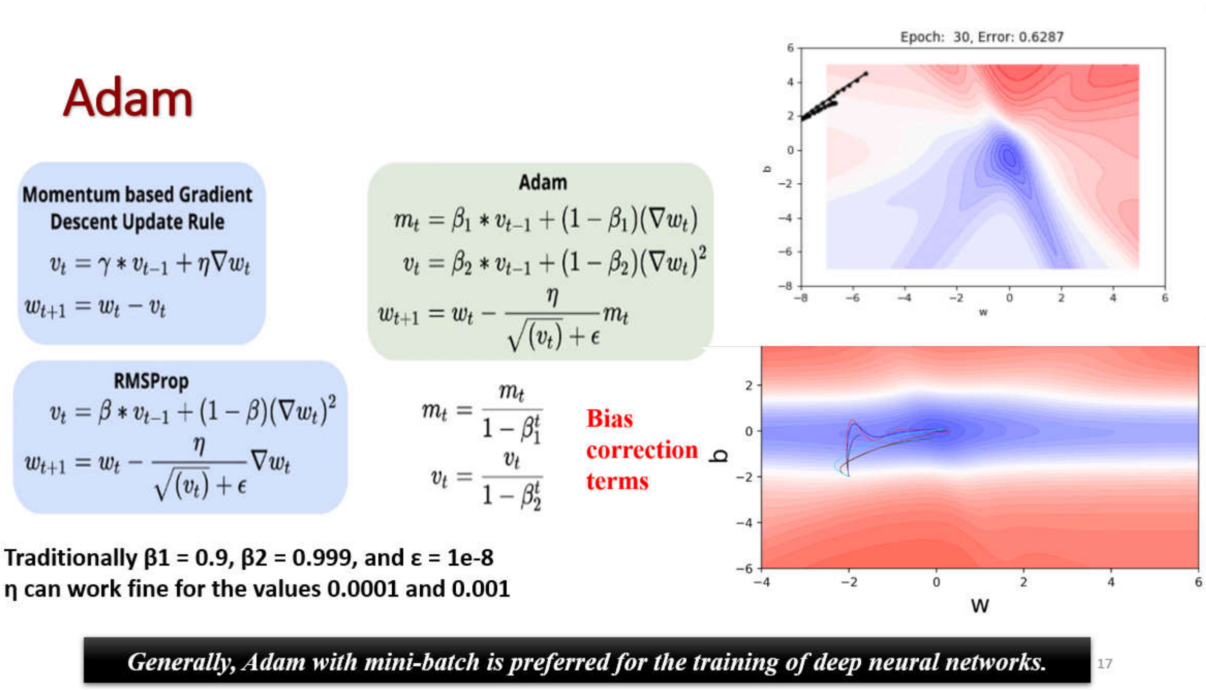
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [43]:
```
def ADAM(x,y,maxEpochs, beta1 , beta2 , learningRate ,convergence , epsilon):
    loss=[]
    thetaList0=[]
    thetaList1=[]
    ypredictedEpochs=[]
    X=np.column_stack((np.ones(len(x),dtype=int),x)) #more columns x0 ,x1
    y=y.reshape(-1,1)     #(shape(20,1))
    m=(X.shape)[0]        #m=20
    thetas=np.zeros((X.shape[1],1))
    count=0
    epoch=1

    v=0
    mt=0
    while epoch < maxEpochs +1:
        count +=1

        ypredicted = X @ thetas # (20,2) @ (2,1) ===> (20,1)
        costOld=(np.sum(np.square(ypredicted - y)))/ (2*m) #Mean Square Error (ol

        Gradient = (np.transpose(X) @ (ypredicted - y) ) / m # (2,20) @ (20,1) ==

        mt = mt/(1-(beta1**epoch))
        v = v / (1-(beta2**epoch))

        mt=(beta1 * mt ) + ((1-beta1) * Gradient)
        v= (beta2 * v) +  ((1-beta2) * (np.square(Gradient)))


        thetas =thetas - ((learningRate * mt) / (np.sqrt(v) + epsilon)) #(2,1)
        thetaList0.append(thetas[0])
        thetaList1.append(thetas[1])


        ypredicted = X @ thetas # (20,2) @ (2,1) ===> (20,1)
        costNew=(np.sum(np.square(ypredicted - y)))/ (2*m) #Mean Square Error (Ne

        loss.append(costNew) #loss list
        ypredictedEpochs.append(ypredicted)

        if abs(costOld - costNew) < convergence:
            print(f'convergence occur after ({count}) iterations')
            return r2_score(y,ypredicted) ,thetas ,ypredicted ,loss ,thetaList0 ,

        epoch+=1

    print(f'sorru=y Max_epochs ({maxEpochs}) have occured')
    return r2_score(y,ypredicted),thetas ,ypredicted ,loss ,thetaList0 ,thetaList
```

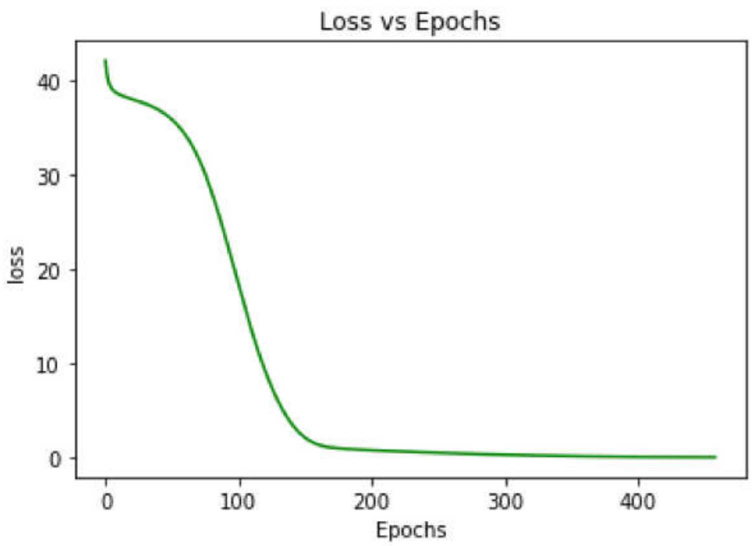
In [44]:
```
R2Score,thetas,ypredicted,loss,thetaList0,thetaList1,ypredictedEpochs=ADAM(x,y,10
```

```
convergence occur after (459) iterations
```

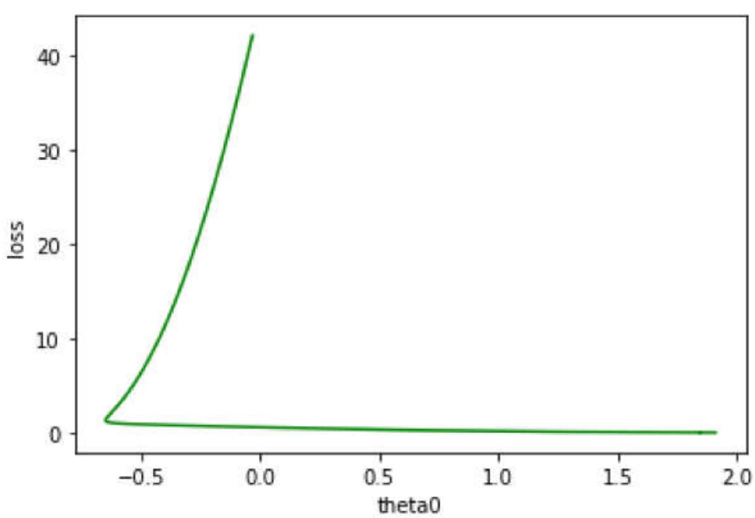
In [45]:
```
R2Score
```
Out[45]:
```
0.9998300068036162
```

In [46]:
```
thetas
```
Out[46]:
```
array([[ 1.90957379],
       [-0.98774668]])
```

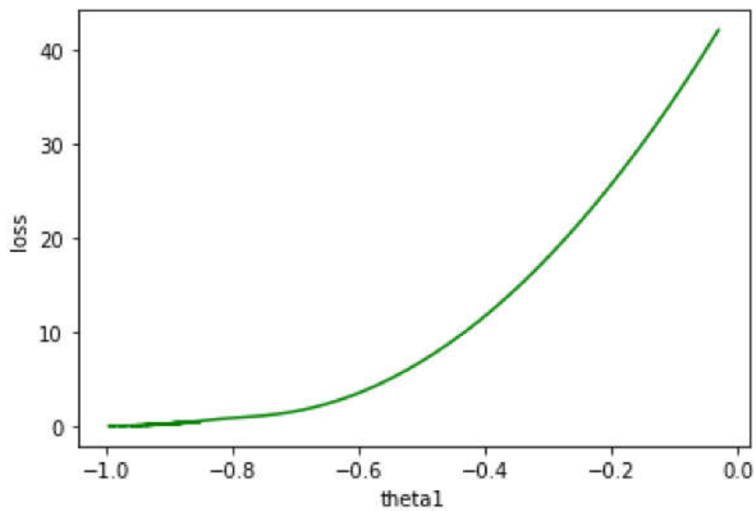
In [47]:
```
plt.plot(loss , color="green")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.title("Loss vs Epochs")
plt.show()
```



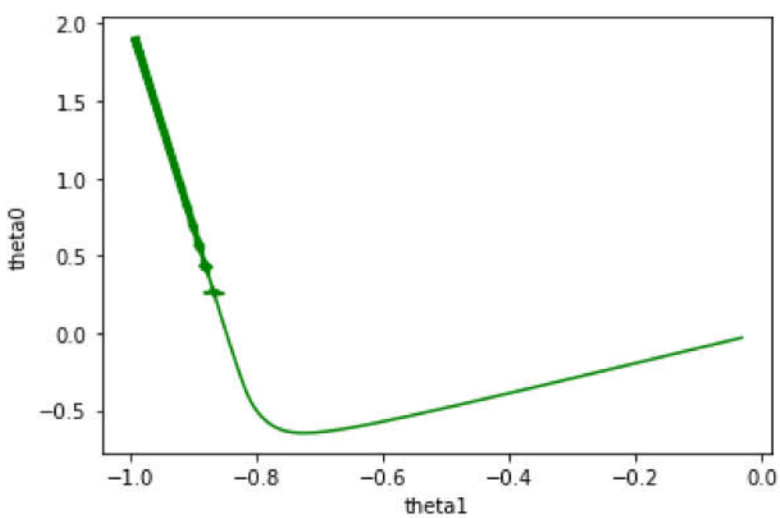
In [48]:
```
plt.plot(thetaList0,loss,color="green")
plt.xlabel("theta0")
plt.ylabel("loss")
plt.show()
```



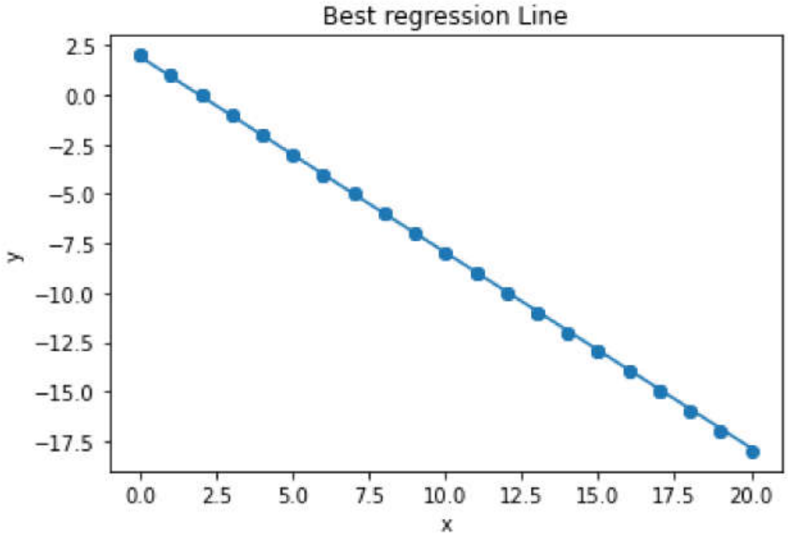
In [49]:
```
plt.plot(thetaList1,loss,color="green")
plt.xlabel("theta1")
plt.ylabel("loss")
plt.show()
```



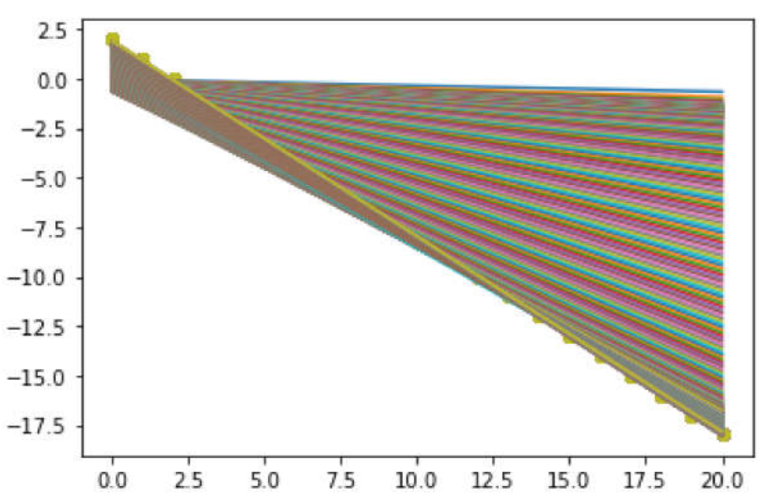
In [50]:
```
plt.plot(thetaList1,thetaList0,color="green")
plt.xlabel("theta1")
plt.ylabel("theta0")
plt.show()
```



In [51]:
```
plt.scatter(x,y)
plt.plot(x,ypredicted)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Best regression Line")
plt.show()
```



In [52]:
```
for h in ypredictedEpochs:
    plt.scatter(x,y)
    plt.plot(x,h)
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: