# Week #3 Assignments – Satish Ramachandran

## Problem #1

'''

Week 3 - Problem 1

Solution using Scikit-learn and Tensorflow

'''

import numpy as np

import matplotlib.pyplot as plt

import tensorflow.compat.v1 as tf

from sklearn import linear\_model

# Disable 2.0 behavior

tf.disable\_v2\_behavior()

### Generate the data ###

def generate\_data(random\_seed, n\_samples):

tf.set\_random\_seed(random\_seed)

train\_x = np.linspace(0,20,n\_samples)

train\_y = 3.7 \* train\_x + 14 + 4 \* np.random.randn(n\_samples)

print("X data")

print("------")

print("Size: " + str(np.shape(train\_x)))

print(train\_x)

print("Y data")

print("------")

print("Size: " + str(np.shape(train\_y)))

print(train\_y)

plt.plot(train\_x, train\_y,'o')

plt.waitforbuttonpress()

plt.close()

return(train\_x, train\_y)

### SciKit Learn method

def scikit\_method(x\_data, y\_data):

print("Using SciKit learn..")

linear\_reg = linear\_model.LinearRegression()

print("Dimensions: X: " + str(x\_data.ndim) + ", Y: " + str(y\_data.ndim))

# IMPORTANT: LinearRegression expects a 2-D array. So, add a dimension using

# reshape()

linear\_reg.fit(x\_data.reshape(-1,1), y\_data.reshape(-1,1))

print("Slope : " + str(linear\_reg.coef\_))

print("Intercept: " + str(linear\_reg.intercept\_))

return (linear\_reg.coef\_, linear\_reg.intercept\_)

### TensorFlow method

def tensorflow\_method(x\_data, y\_data, learn\_rate, epochs):

print("Tensor flow method..")

graph = tf.Graph()

with graph.as\_default():

slope = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

intercept = tf.Variable(tf.zeros([1]))

response = slope\*x\_data + intercept

cost = tf.reduce\_mean(tf.square(response - y\_data))

optimizer = tf.train.GradientDescentOptimizer(learn\_rate).minimize(cost)

with tf.Session(graph=graph) as session:

init = tf.global\_variables\_initializer()

session.run(init)

for epoch in range(epochs):

session.run(optimizer)

if ( epoch % 1000 ) == 0:

print("Plot after " + str(epoch) + " iterations")

plt.plot(x\_data, y\_data, 'o', label = 'step = {}'.format(epoch))

plt.plot(x\_data, session.run(slope)\*x\_data + session.run(intercept))

plt.legend()

plt.show()

plt.waitforbuttonpress()

print("Slope = ",session.run(slope))

print("Intercept = ",session.run(intercept))

#return(tf.cast(session.run(slope), tf.int32), tf.cast(session.run(intercept), tf.int32))

return(session.run(slope), session.run(intercept))

train\_x, train\_y = generate\_data(42, 30)

s\_slope, s\_intercept = scikit\_method(train\_x, train\_y)

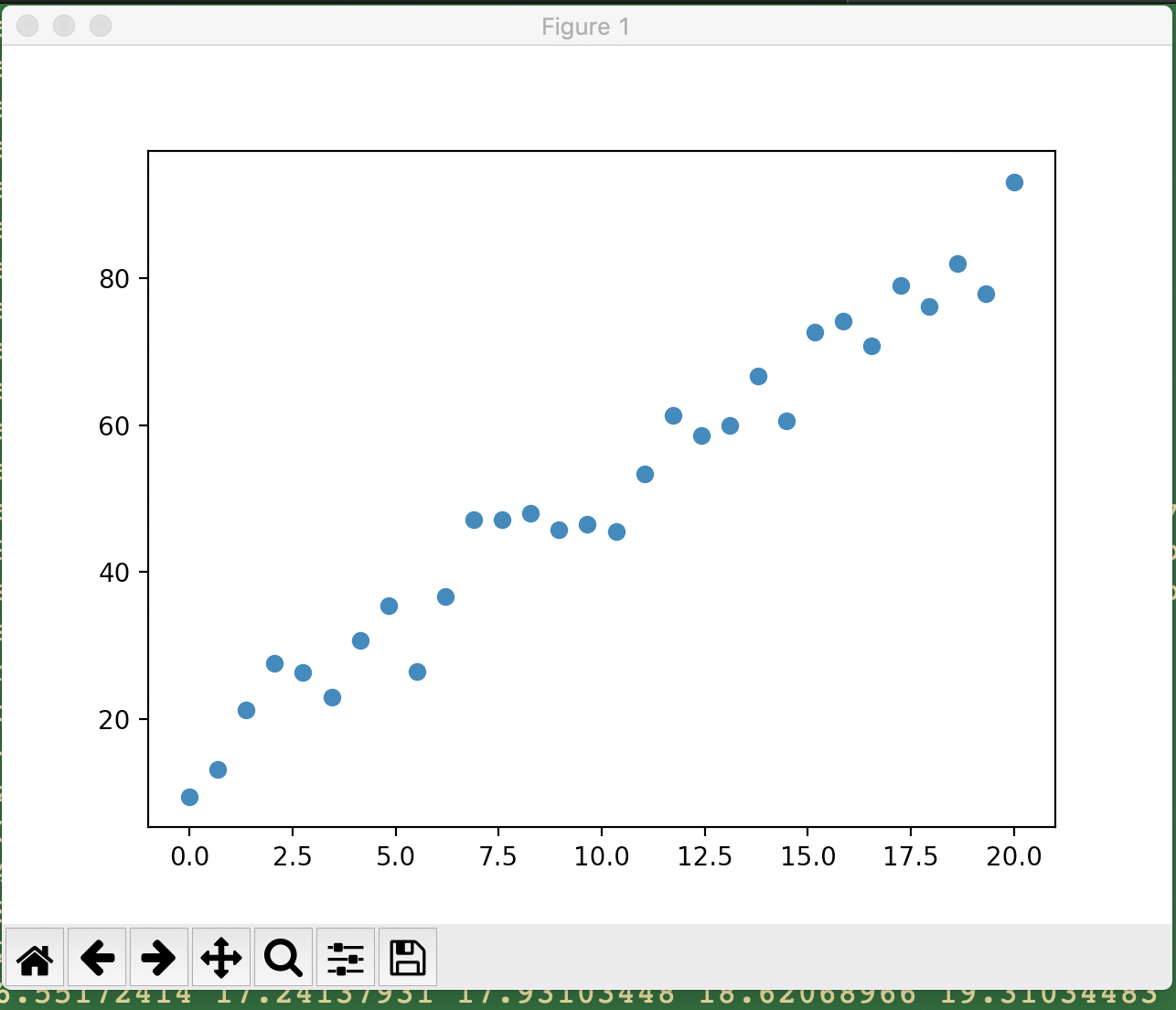
t\_slope, t\_intercept = tensorflow\_method(train\_x, train\_y, 0.001, 10000)

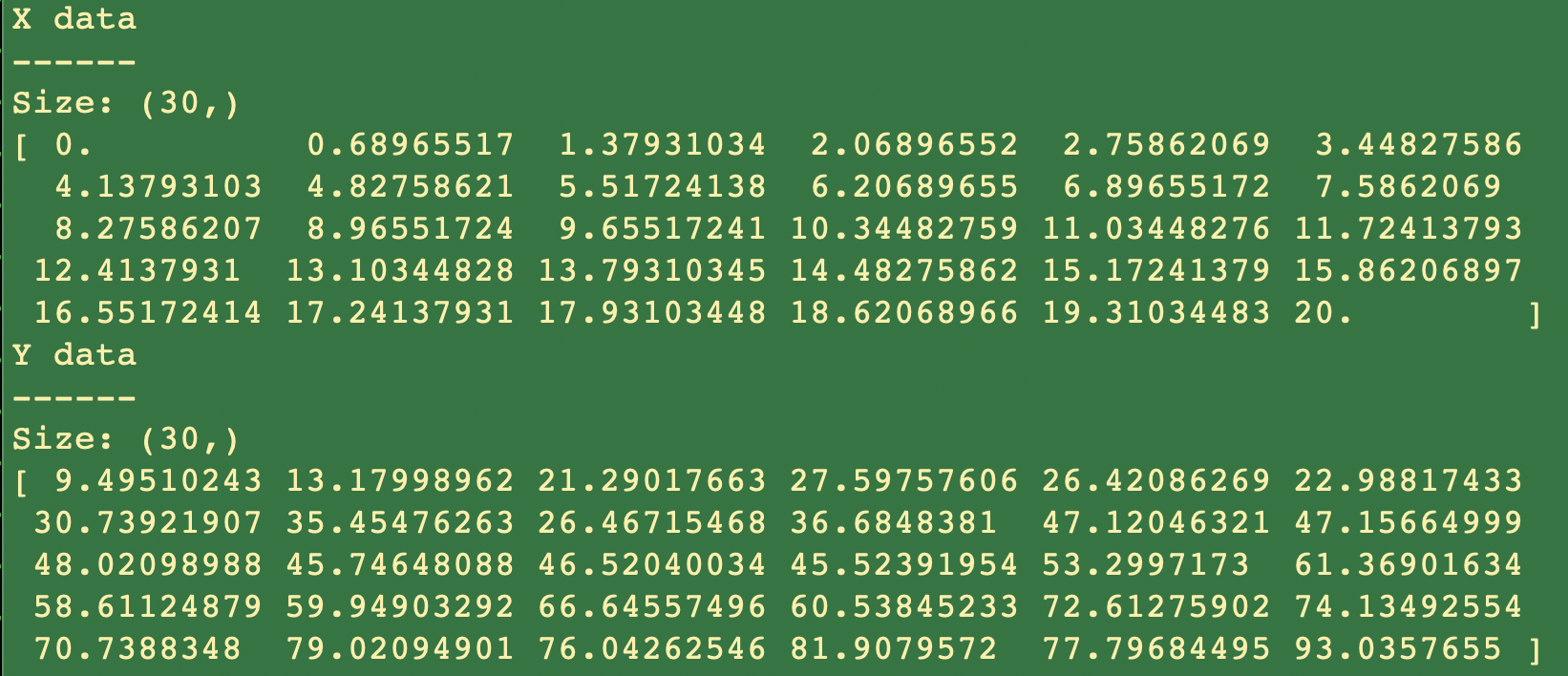
print("RESULTS" + "\n" + "-------")

print("SciKit Learn : slope: " + str(s\_slope) + " intecept: " + str(s\_intercept))

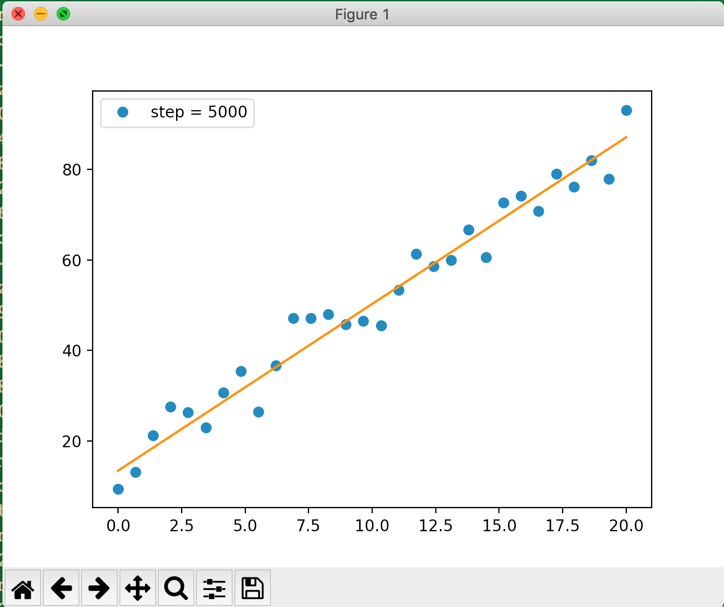
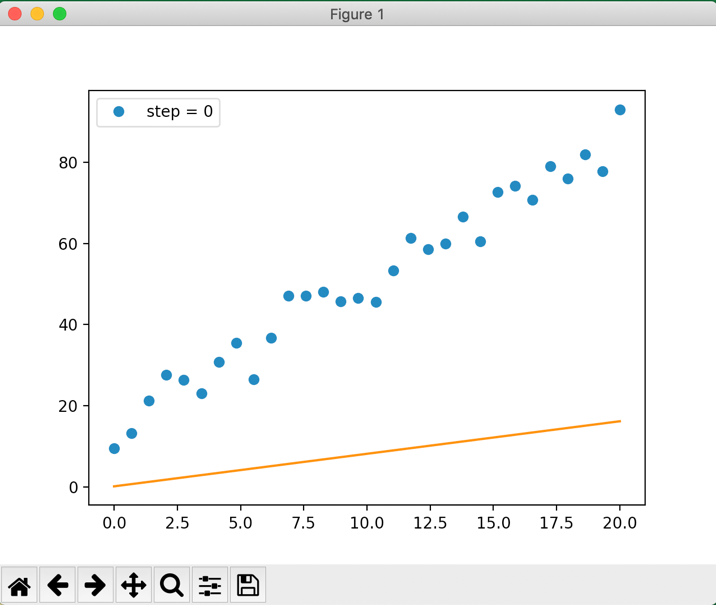
print("TensorFlow : slope: " + str(t\_slope) + " intecept: " + str(t\_intercept))

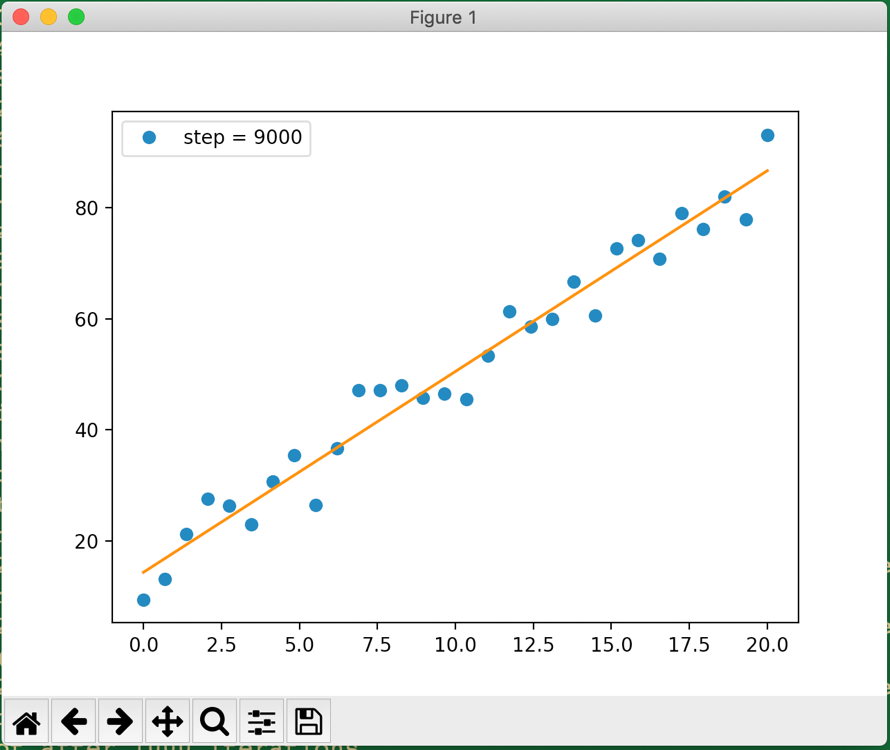
This is the plot of the generated data set

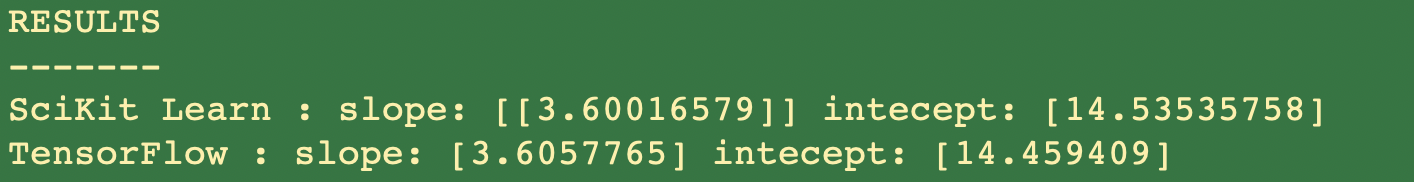




Tensor flow plots to start with, and after a few thousand epochs:







## Problem 2

'''

Week 3 - Problem 2

Solution using Keras

'''

import numpy as np

import matplotlib.pyplot as plt

import tensorflow.compat.v1 as tf

import keras

from keras.models import Sequential

from keras.layers import Dense

# Disable 2.0 behavior

tf.disable\_v2\_behavior()

### Generate the data ###

def generate\_data(random\_seed, n\_samples):

tf.set\_random\_seed(random\_seed)

train\_x = np.linspace(0,20,n\_samples)

train\_y = 3.7 \* train\_x + 14 + 4 \* np.random.randn(n\_samples)

print("X data")

print("------")

print("Size: " + str(np.shape(train\_x)))

print(train\_x)

print("Y data")

print("------")

print("Size: " + str(np.shape(train\_y)))

print(train\_y)

plt.plot(train\_x, train\_y,'o')

plt.waitforbuttonpress()

plt.close()

return(train\_x, train\_y)

def model\_keras(x\_data, y\_data, epochs):

model = Sequential()

model.add(Dense(1, input\_dim=1, kernel\_initializer='normal', activation='linear'))

#Compile the model

model.compile(loss='mean\_squared\_error', optimizer='rmsprop', metrics=['mse'])

#Dump the model

model.summary()

#Suppressing the per-epoch messages

hist = model.fit(x\_data, y\_data, epochs=epochs, verbose=0)

weightBias = model.layers[0].get\_weights()

#print('Weight and Bias with Keras: " + weightBias)

print(weightBias)

plt.plot(train\_x, train\_y,'o')

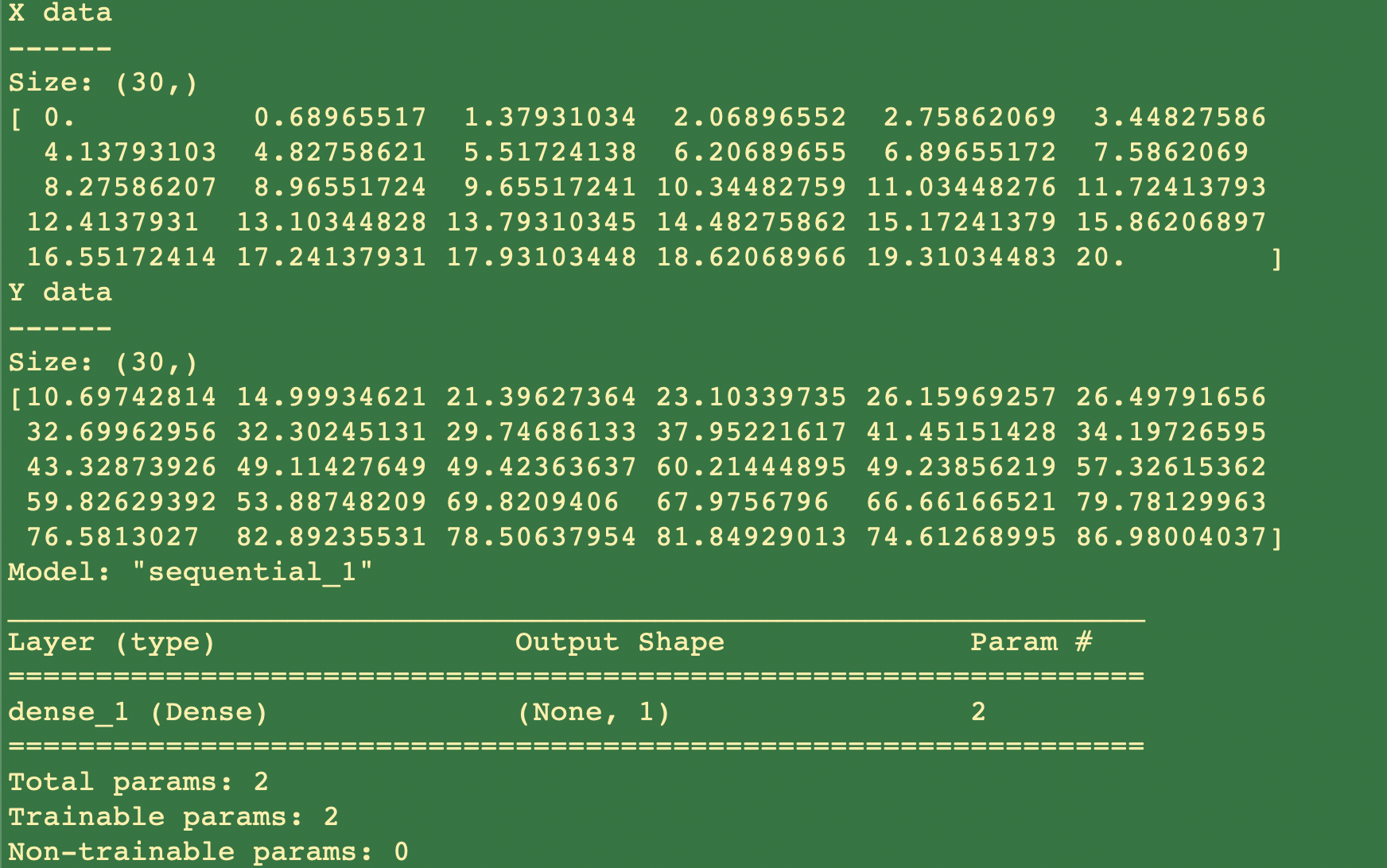
plt.plot(x\_data, weightBias[0][0]\*x\_data + weightBias[1])

plt.waitforbuttonpress()

train\_x, train\_y = generate\_data(42, 30)

model\_keras(train\_x, train\_y, 20000)

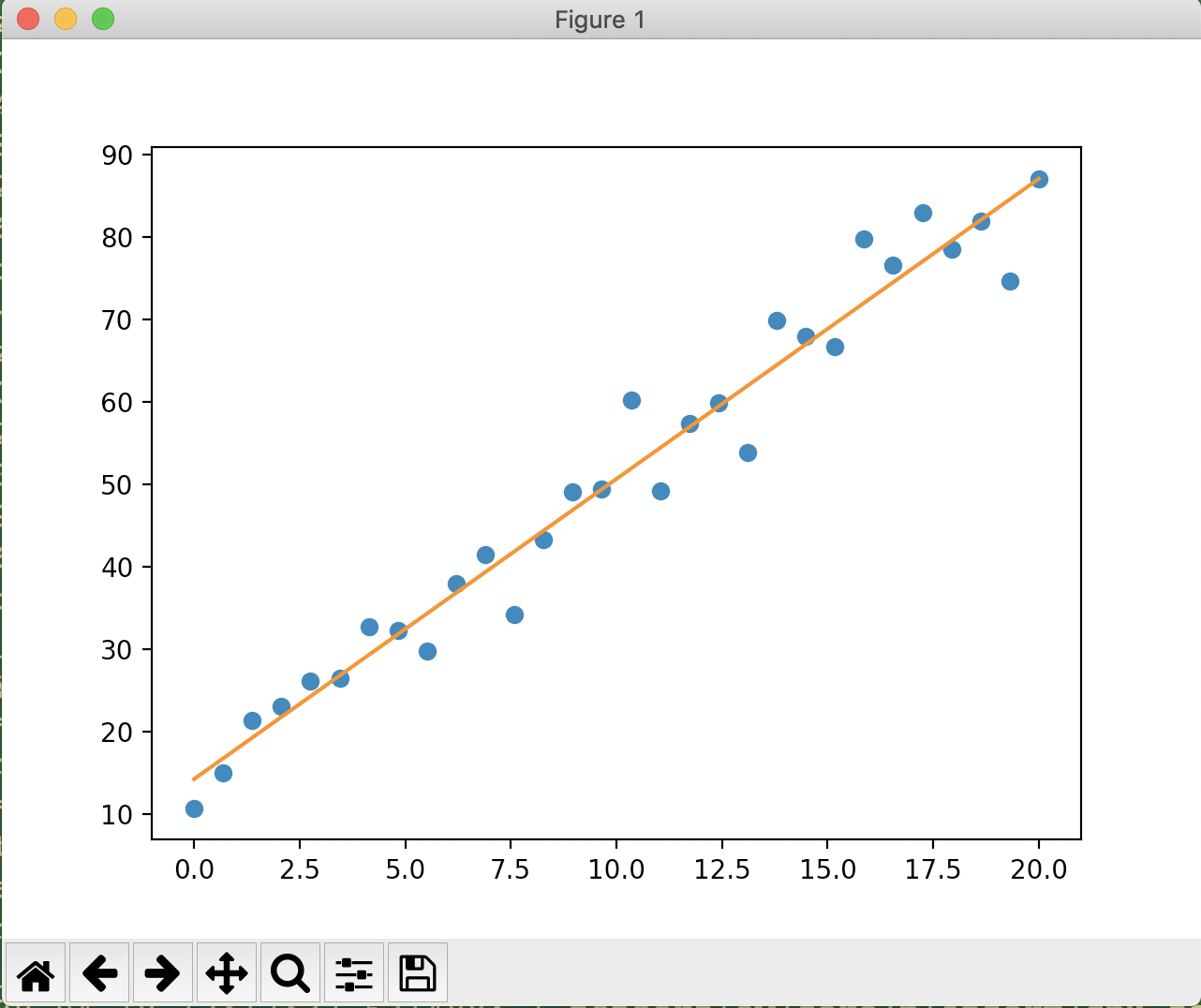
**Generated test data:**



**Model output for slope and intercept with Keras:**



This is the plot of the samples generated and the final line obtained after keras training



**It can be seen that the values are consistent across SciKit learn, TensorFlow and Keras**

## Problem 3

'''

Week 3 - Problem 3

Solution using Scikit-learn and Tensorflow

Housing data in a CSV file

'''

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import tensorflow as tf

from sklearn import linear\_model

from sklearn import preprocessing

import pandas as pd

def plot\_plane(predictors, target, b1, b2, intercept):

plt.clf()

figure = plt.figure()

x1\_surf, x2\_surf = np.meshgrid(np.linspace(predictors[:,0].min(), predictors[:,0].max(), 500), np.linspace(predictors[:,1].min(), predictors[:,1].max(), 500))

y\_surf = x1\_surf\*b1 + x2\_surf\*b2 + intercept

ax = Axes3D(figure)

ax.scatter(predictors[:,0], predictors[:,1], target, c='blue', alpha=0.5)

plt\_surface = ax.plot\_surface(x1\_surf, x2\_surf, y\_surf, color='red', alpha=0.2)

ax.set\_xlabel('Bedrooms')

ax.set\_ylabel('Sqft')

ax.set\_zlabel('Price')

plt.show()

plt.waitforbuttonpress()

### Import the data from the CSV file ###

def extract\_predictor\_target():

data = pd.read\_csv('kc\_house\_data.csv')

print(data.head())

predictors = preprocessing.minmax\_scale(data[['bedrooms','sqft\_living']])

target = preprocessing.minmax\_scale(data[['price']])

#plot\_data(predictors, target)

print(predictors.shape)

print(target.shape)

return (predictors, target)

### SciKit Learn method

def scikit\_method(predictors, target):

print("Using SciKit learn..")

linear\_reg = linear\_model.LinearRegression()

#print("Dimensions: X: " + str(predictors.ndim) + ", Y: " + str(target.ndim))

# IMPORTANT: LinearRegression expects a 2-D array. So, add a dimension using

# reshape()

#linear\_reg.fit(x\_data.reshape(-1,1), y\_data.reshape(-1,1))

linear\_reg.fit(predictors, target)

print("Slope : " + str(linear\_reg.coef\_))

print("Intercept: " + str(linear\_reg.intercept\_))

return (linear\_reg.coef\_, linear\_reg.intercept\_)

### TensorFlow method

def tensorflow\_method(x\_data, y\_data, learn\_rate, epochs):

print("Tensor flow method..")

graph = tf.Graph()

with graph.as\_default():

slope = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

x1 = tf.placeholder(dtype=np.float32)

x2 = tf.placeholder(dtype=np.float32)

y = tf.placeholder(dtype=np.float32)

#w1 = tf.Variable([0], dtype=np.float32, name="weight1")

#w2 = tf.Variable([0], dtype=np.float32, name="weight2")

w1 = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

w2 = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

b = tf.Variable([0], dtype=np.float32, name="bias")

response = w1\*x1 + w2\*x2 + b

cost = tf.reduce\_mean(tf.square(response - y))

optimizer = tf.train.GradientDescentOptimizer(learn\_rate).minimize(cost)

with tf.Session(graph=graph) as session:

init = tf.global\_variables\_initializer()

session.run(init)

x1\_list = x\_data[:,0].tolist()

x2\_list = x\_data[:,1].tolist()

y\_list = y\_data[:,0].tolist()

print(x1\_list[0:6])

print(x2\_list[0:6])

print(y\_list[0:6])

for epoch in range(epochs):

session.run(optimizer, {x1:x1\_list, x2:x2\_list, y:y\_list})

if (epoch % 10000) == 0:

print("w1 = ",session.run(w1))

print("w2 = ",session.run(w2))

print("b = ",session.run(b))

return(session.run(w1), session.run(w2), session.run(b))

predictors, target = extract\_predictor\_target()

print(predictors)

print(target)

print(predictors[:,0], predictors[:,1], target[:,0])

#Get the linear regression using SciKit Learn

coef, intercept = scikit\_method(predictors, target)

print(coef[0][0])

print(coef[0][1])

print(intercept[0])

plot\_plane(predictors, target, coef[0][0], coef[0][1], intercept[0])

w1, w2, b = tensorflow\_method(predictors, target, 0.008, 1000000)

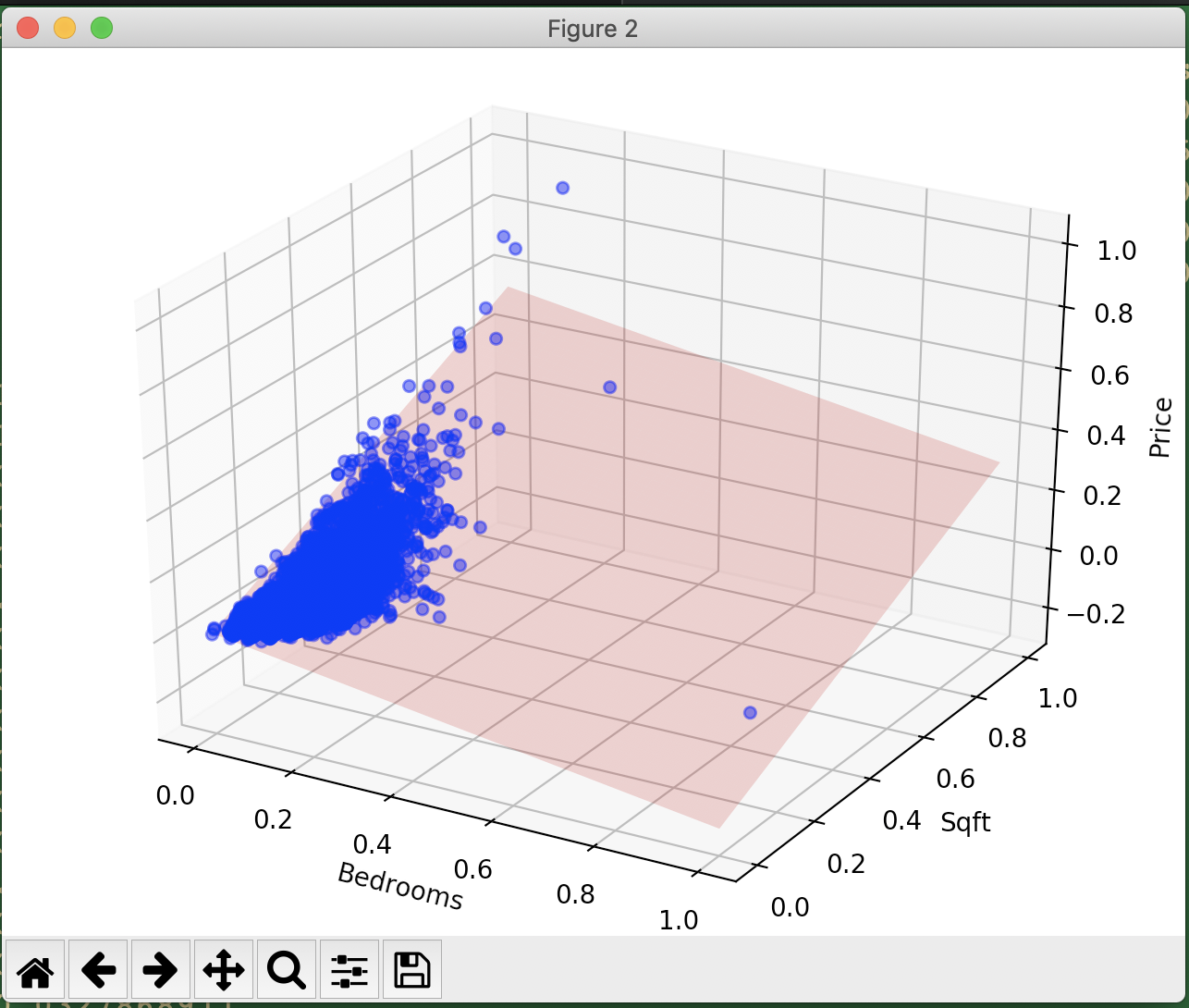
#plot\_plane(predictors, target, w1, w2, b)

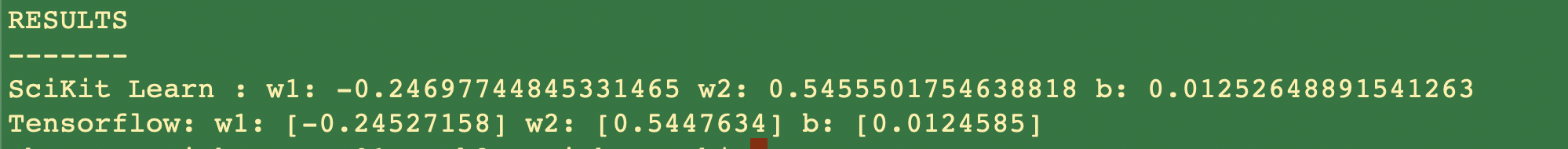
print("RESULTS" + "\n" + "-------")

print("SciKit Learn : w1: " + str(coef[0][0]) + " w2: " + str(coef[0][1]) + " b: " + str(intercept[0]))

print("Tensorflow: w1: " + str(w1) + " w2: " + str(w2) + " b: " + str(b))

**Solution plot**





**The SciKit learn values and TensorFlow values are extremely close.**

## Problem 4

'''

Week 3 - Problem 4

Solution using Keras

'''

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import keras

from keras.models import Sequential

from keras.layers import Dense

from sklearn import preprocessing

# Extract data from the CSV file

def extract\_predictor\_target():

data = pd.read\_csv('kc\_house\_data.csv')

print(data.head())

predictors = preprocessing.minmax\_scale(data[['bedrooms','sqft\_living']])

target = preprocessing.minmax\_scale(data[['price']])

#plot\_data(predictors, target)

print(predictors.ndim)

print(target.ndim)

return (predictors, target)

# Keras model definition and execution

def model\_keras(x\_data, y\_data, epochs):

model = Sequential()

model.add(Dense(1, input\_dim=2, kernel\_initializer='normal', activation='linear'))

#Compile the model

model.compile(loss='mean\_squared\_error', optimizer='rmsprop', metrics=['mse'])

#Dump the model

model.summary()

#Suppressing the per-epoch messages

hist = model.fit(x\_data, y\_data, epochs=epochs, verbose=0)

weightBias = model.layers[0].get\_weights()

return weightBias

predictors, target = extract\_predictor\_target()

print('Predictor shape ', predictors.shape)

print('Target shape ', target.shape)

#Convert to arrays

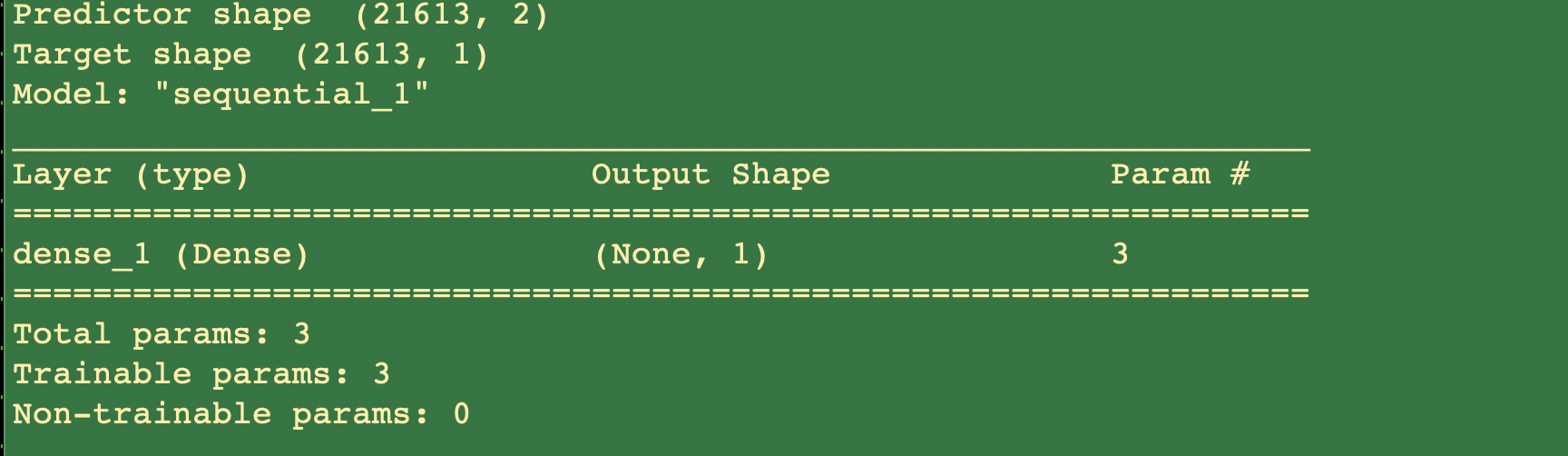
pred\_array = np.array(predictors)

target\_array = np.array(target)

weightBias = model\_keras(pred\_array, target\_array, 100)

print("RESULT:" + "\n" + "-------")

print("w1: " + str(weightBias[0][0]) + " w2: " + str(weightBias[0][1]) + " bias: " + str(weightBias[1]))





**It can be seen that the values are consistent across SciKit learn, TensorFlow and Keras**