LAB 3

PART A: Prerequisite for Linear Regression implementation

1. Create an array x = [1, 1, 2, 3, 4, 3, 4, 6, 4] using numpy. Calculate a function h(x)=t0+t1*x, where t0=1.2 and t1=0.5, for all values of x and plot a graph with x on one axis and h(x)on another axis.

In [28]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
plt.style.use("Solarize_Light2")
import random
np.random.seed(9)
np.set_printoptions(suppress=True)
```

In [29]:

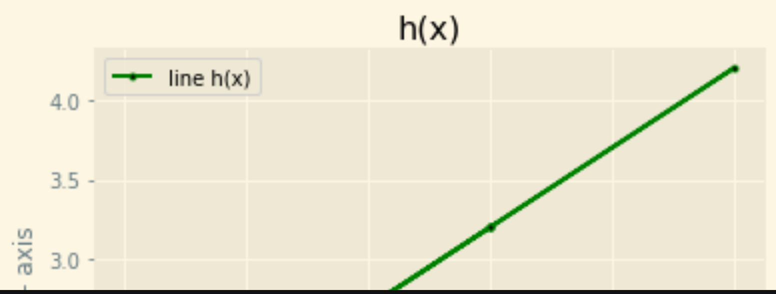
```
x = [1, 1, 2, 3, 4, 3, 4, 6, 4]
def h(x, t0 = 1.2, t1 = 0.5):
    return t0 + (t1 * x)


print(h(x[5]))
```

```
2.7
```

In [30]:

```
y = [h(i) for i in x]
print(x)
print(y)
plt.plot(x, y, color='green', label = "line h(x)", marker='o',
markerfacecolor='black', markersize=3)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('h(x)')
plt.legend()
plt.show()
```

```
[1, 1, 2, 3, 4, 3, 4, 6, 4]
[1.7, 1.7, 2.2, 2.7, 3.2, 2.7, 3.2, 4.2, 3.2]
```



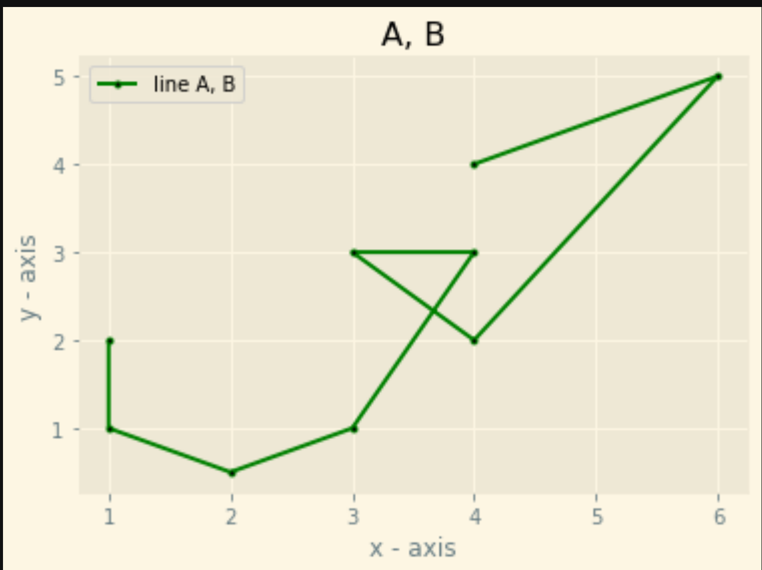
2. Create two arrays A and B with the following values using numpy array. Let (Ai,Bi) represent a data point with i th element of A and B. A = [1, 1, 2, 3, 4, 3, 4, 6, 4] B = [2, 1, 0.5, 1, 3, 3, 2, 5, 4] Find out the dot product of the vectors. [Hint use numpy np.dot(a,b)]

In [31]:
```
A = [1, 1, 2, 3, 4, 3, 4, 6, 4]
B = [2, 1, 0.5, 1, 3, 3, 2, 5, 4]
print("Dot Product of the Vectors: ", np.dot(A, B)) # 2+1+1+3+12+9+8+30+16
= 82
```

```
Dot Product of the Vectors:  82.0
```

3. Plot a graph marking the data points (Ai,Bi) with A on the X-axis and B on the Y-axis.

In [32]:
```
plt.plot(A, B, color='green', label = "line A, B", marker='o',
markerfacecolor='black', markersize=3)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('A, B')
plt.legend()
plt.show()
```



4. Calculate Mean Square Error (MSE) of A and B with the formulae where n is the number of sample data points.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(A^i - B^i)^2$$

In [33]:

```
print("mean_squared_error = ", mean_squared_error(A, B)) #using in-built function
sum = 0
for i in range(len(A)):
    sum = sum + ((A[i]-B[i]) * (A[i]-B[i]))
mse = sum/len(A)
print("mean_squared_error = ", mse) #using formula
```

```
mean_squared_error =  1.4722222222222223
mean_squared_error =  1.4722222222222223
```

5. Modify the above equation with the following cost function. Implement as a function with prototype def compute_cost_function(n,t1,A,B):

Take h(x) =t1*x and t1= 0.5 Modify the above code iterating for different values of t1 and calculate J(t1).Try with t1 =0.1,0.3,0.5,0.7,0.8. Plot a graph with t1 on X-axis and J(t1) on Y-axis. [hint sum_squared_error = np.square(np.dot(features, theta) - values).sum() cost = sum_squared_error / (2*m)]

$$J(t_1) = \frac{1}{2n}\sum_{i=1}^{n}(h(A^i) - B^i))^2$$

In [34]:

```
def compute_cost_function(n, t1, A, B):
    sum_squared_error = np.square(np.dot(A, t1) - B).sum()
    cost = sum_squared_error / (2*n)
    return cost
```

In [35]:

```
t1 =[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
print("Cost Function = ", compute_cost_function(len(A), t1, A, B))
```

```
Cost Function =  109.20222222222225
```

PART B : Linear Regression Implementation

1. Linear regression with one variable.

a. Generate a new data set from student scores with one feature studytime and output variable average grade = (G1+G2+G3)/3

In [36]:

```
df_student = pd.read_csv("datasets_52721_99691_student-mat.csv")
df_student
```

Out[36]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
390	MS	M	20	U	LE3	A	2	2	services	services	...	5	5	4
391	MS	M	17	U	LE3	T	3	1	services	services	...	2	4	5
392	MS	M	21	R	GT3	T	1	1	other	other	...	5	5	3
393	MS	M	18	R	LE3	T	3	2	services	other	...	4	4	1
394	MS	M	19	U	LE3	T	1	1	other	at_home	...	3	2	3

In [37]:

```
df_student["average_grade"] = ( df_student["G1"] + df_student["G2"] + df_student["G3"] ) / 3
```

b. Load the new data set

In [38]:

```
df_student
```

Out[38]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	freetime	goout	Dalc	Wa
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	3	4	1	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	3	3	1	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	3	2	2	
3	GP	F	15	U	GT3	T	4	2	health	services	...	2	2	1	
4	GP	F	16	U	GT3	T	3	3	other	other	...	3	2	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
390	MS	M	20	U	LE3	A	2	2	services	services	...	5	4	4	
391	MS	M	17	U	LE3	T	3	1	services	services	...	4	5	3	
392	MS	M	21	R	GT3	T	1	1	other	other	...	5	3	3	
393	MS	M	18	R	LE3	T	3	2	services	other	...	4	1	3	
394	MS	M	19	U	LE3	T	1	1	other	at_home	...	2	3	3	

395 rows × 34 columns

In [39]:

```
df_student.columns
```

Out[39]:

```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'G1', 'G2', 'G3', 'average_grade'],
      dtype='object')
```

In [40]:

```
X = df_student[['studytime']]
y = df_student[['average_grade']]
X
```

Out[40]:

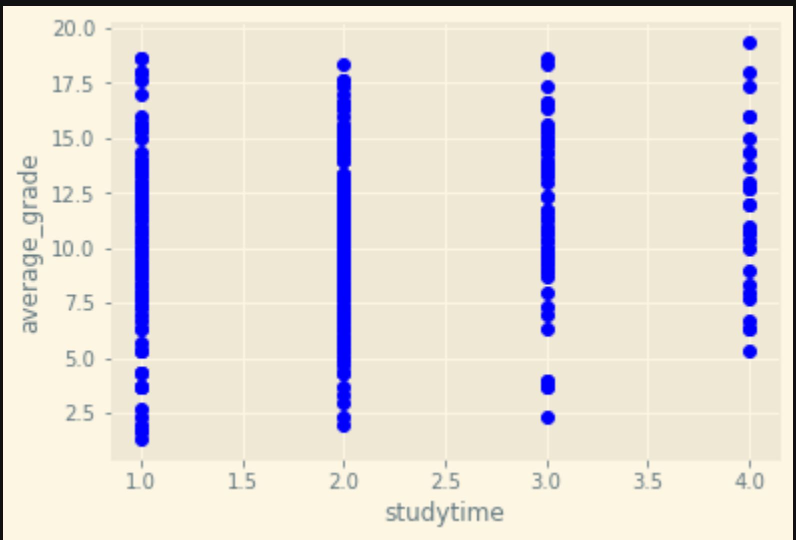
	studytime

	studytime
0	2
1	2
2	2
3	3
4	2
...	...
390	2
391	1
392	1
393	1
394	1

c. Plot data

In [41]:

```
plt.scatter(X, y, c ="blue")
plt.xlabel("studytime")
plt.ylabel("average_grade")
plt.show()
```



d. Implement linear regression using inbuilt package python Scikit

In [42]:

```
y
```

Out[42]:

	average_grade
0	5.666667
1	5.333333
2	8.333333
3	14.666667
4	8.666667
...	...
390	9.000000

	average_grade
391	15.333333
392	8.333333
393	11.000000
394	8.666667

In [43]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
shuffle = False)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
y_pred = [i[0] for i in y_pred]
print(y_pred[0:10])
print(y_test[0:10])
```

```
[10.647813657332208, 10.647813657332208, 11.400656227665518, 10.27139237216555, 10.6478
13657332208, 10.27139237216555, 11.024234942498863, 11.400656227665518, 10.647813657332
208, 10.647813657332208]
     average_grade
296       6.333333
297       8.666667
298      13.666667
299      15.666667
300      11.000000
301      10.666667
302      13.666667
303      17.333333
304      14.000000
305      12.666667
```

e. Implement gradient descent algorithm with the function prototype def gradient_descent(alpha, x, y, max_iter=1500): where alpha is the learning rate, x is the input feature vector. y is the target. Subject the feature vector to normalisation step if needed. Convergence criteria: when no: of iterations exceed max_iter.

[hint sum_squared_error = np.square(np.dot(features, theta) - values).sum()
cost = sum_squared_error / (2*m)]

In [44]:
```
x = df_student['studytime']
y = df_student['average_grade']

x = (x - x.mean()) / x.std()
x = np.c_[np.ones(x.shape[0]), x]
```

In [45]:

```
def gradient_descent(x, y, theta, iterations, alpha):
    past_costs = []
    past_thetas = [theta]
    for i in range(iterations):
        prediction = np.dot(x, theta)
        error = prediction - y
        cost = 1/(2*m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)

    return past_thetas, past_costs
```

f. Vary learning rate from 0.1 to 0.9 and observe the learned parameter.

In [46]:

```
alpha = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] #Learning rate
iterations = 1500 #No. of iterations
m = y.size #No. of data points
np.random.seed(123) #Set the seed
theta = np.random.rand(2) #Pick some random values to start with
for i in range(len(alpha)):
    past_thetas, past_costs = gradient_descent(x, y, theta, iterations, 
alpha[i])
    theta = past_thetas[-1]
    print("\n\nLearning Rate = ", alpha[i])
    print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))
    print("Cost = ", past_costs[-1])
```

```


Learning Rate =  0.1
Gradient Descent: 10.68, 0.50
Cost =  6.69239454434365


Learning Rate =  0.2
Gradient Descent: 10.68, 0.50
Cost =  6.692394544343652


Learning Rate =  0.3
Gradient Descent: 10.68, 0.50
Cost =  6.69239454434365


Learning Rate =  0.4
Gradient Descent: 10.68, 0.50
Cost =  6.69239454434365


Learning Rate =  0.5
Gradient Descent: 10.68, 0.50
Cost =  6.69239454434365


Learning Rate =  0.6
Gradient Descent: 10.68, 0.50
```

```
Cost =  6.69239454434365


Learning Rate =  0.7
Gradient Descent: 10.68, 0.50
Cost =  6.69239454434365


Learning Rate =  0.8
Gradient Descent: 10.68, 0.50
Cost =  6.69239454434365


Learning Rate =  0.9
Gradient Descent: 10.68, 0.50
```

In [47]:

```
'''
def  cal_cost(theta,X,y):
    m = len(y)
    predictions = X.dot(theta)
    cost = (1/2*m) * np.sum(np.square(predictions-y))
    return cost

def gradient_descent(alpha, X, y, iterations):
    m = len(y)
    theta = np.random.randn(2,1)
    cost_history = np.zeros(iterations)
    theta_history = np.zeros((iterations,2))
    for it in range(iterations):
        prediction = np.dot(X, theta)
        theta = theta -(1/m)*alpha*( X.T.dot((prediction - y)))
        theta_history[it,:] =theta.T
        cost_history[it]  = cal_cost(theta,X,y)

    return cost_history

X = df_student[['studytime']]
y = df_student[['average_grade']]
alpha = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
X_b = np.c_[np.ones((len(X),1)),X]
MSE = []
for learning_rate in alpha:
    cost_history = gradient_descent(learning_rate, X_b, y, 1500)
    MSE.append(cost_history[-1])
for i in range(len(alpha)):
    print("For Learning Rate = ", alpha[i],", MSE = ", MSE[i])
'''
```

Out[47]:

```
'\ndef  cal_cost(theta,X,y):\n    m = len(y)\n    predictions = X.dot(theta)\n    cost
= (1/2*m) * np.sum(np.square(predictions-y))\n    return cost\n\ndef gradient_descent(a
lpha, X, y, iterations):\n    m = len(y)\n    theta = np.random.randn(2,1)\n    cost_hi
story = np.zeros(iterations)\n    theta_history = np.zeros((iterations,2))\n    for it
in range(iterations):\n        prediction = np.dot(X, theta)\n        theta = theta -(1
```

```
/m)*alpha*( X.T.dot((prediction - y)))\n        theta_history[it,:] =theta.T\n        c
ost_history[it]  = cal_cost(theta,X,y)\n        \n    return cost_history  \n\nX = df_s
tudent[[\'studytime\']]\ny = df_student[[\'average_grade\']]\nalpha = [0.1, 0.2, 0.3, 
0.4, 0.5, 0.6, 0.7, 0.8, 0.9]\nX_b = np.c_[np.ones((len(X),1)),X]\nMSE = []\nfor learni
ng_rate in alpha:\n    cost_history = gradient_descent(learning_rate, X_b, y, 1500)\n   
MSE.append(cost_history[-1])\nfor i in range(len(alpha)):\n    print("For Learning Rate
```