Linear Regression

The idea is to write a python program or python notebook which will implement linear regression on a dataset.

Part1.

First download the dataset: data_linregr.csv

Recommended is to use the pandas library to do this:

```
import pandas as pd
import matplotlib.pyplot as plt
import os

filepath = os.getcwd() + "/data/data_linregr.csv" #please
modify to your situation
data = pd.read_csv(filepath, header=None, names =
['mass','acceleration'])
```

Print some rows of this dataset (use the head function of pandas: data.head())

part2

Very interesting part of pandas is that you get all kind of information "for free" on your dataset:

Try the describe() function of pandas and print the result.

Plot the data using matplotlib. Use a 'scatter' plot and use titles on the axis.

part3

Now we will come to the center of the problem, but I assume you are already familiar with much of the theory. However, a basic summary is given below.

The line is given by a linear model:

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Note the matrix notation at the start!

The error to be minimized is often called a cost function. We try to find a optimum value for
θ_0 and θ_1 using a gradient descent method which minimizes this error.
The error (or 'cost') function is given by

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h\theta(x) - y)^2$$

The update rate for each set of coefficients is called alpha. We can take a value of 0.01 for this.
So the cost function can be written mathematically as:

$$\theta_j = \theta_j - \alpha \frac{1}{2m} \sum_{i=1}^{m} (h\theta(x) - y)x_j$$

We simultaneously update \theta_j in each step of a loop.

Now the Python part. A few tips:
1. start with the 2 matrices containing the values of our dataset.

```
x = np.matrix(x.values)
y = np.matrix(y.values)
theta = np.matrix(np.array([0,0])
```

Numpy has an incredible simple operation for transposing a matrix: when a is a matrix a.T is the
transposed version!)
Write a linregCost(x,y,theta) function which will do one approximation using the formula above.
Next, write a gradientDescent(x,y,theta, alpha, iterations) function which will do the gradient
descent approximation. It's nice to plot each approximation and if it starts to make sense only
plot the final result.
Nice extra: you can try to plot the gradient descent result as a contour plot.
Part 4

As you can see in the equation for the hypothesis above: it's written as a transposed matrix. This means that we could easily extend the whole procedure to create a polynomial equation for a dataset.

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

Interesting extra: the alpa learning rate can be adjusted. Try it with some different values.