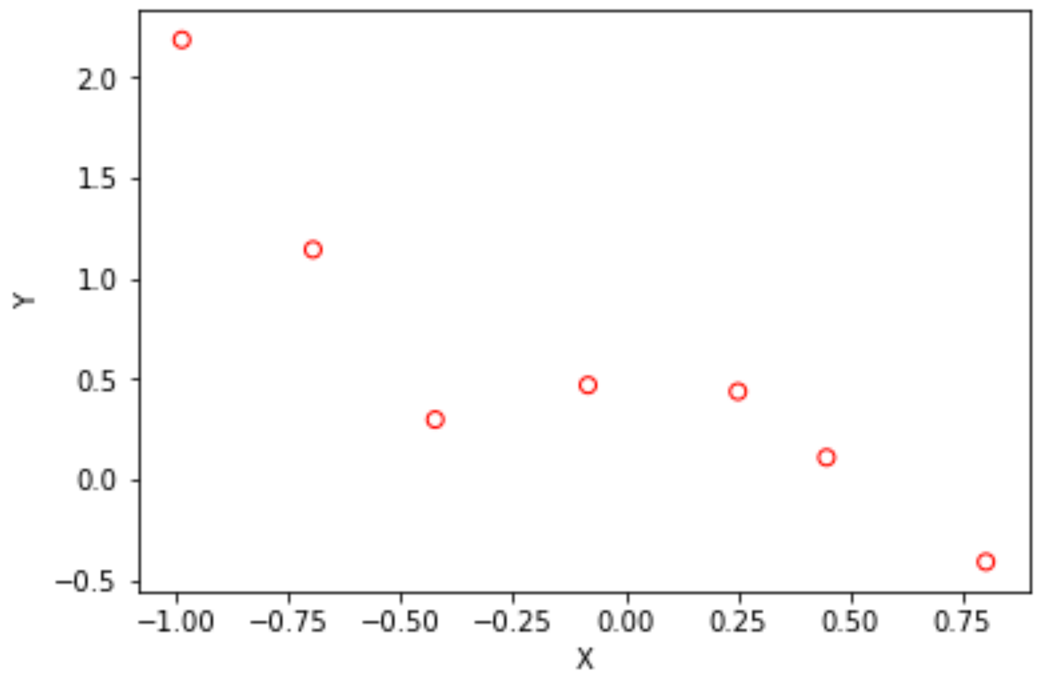
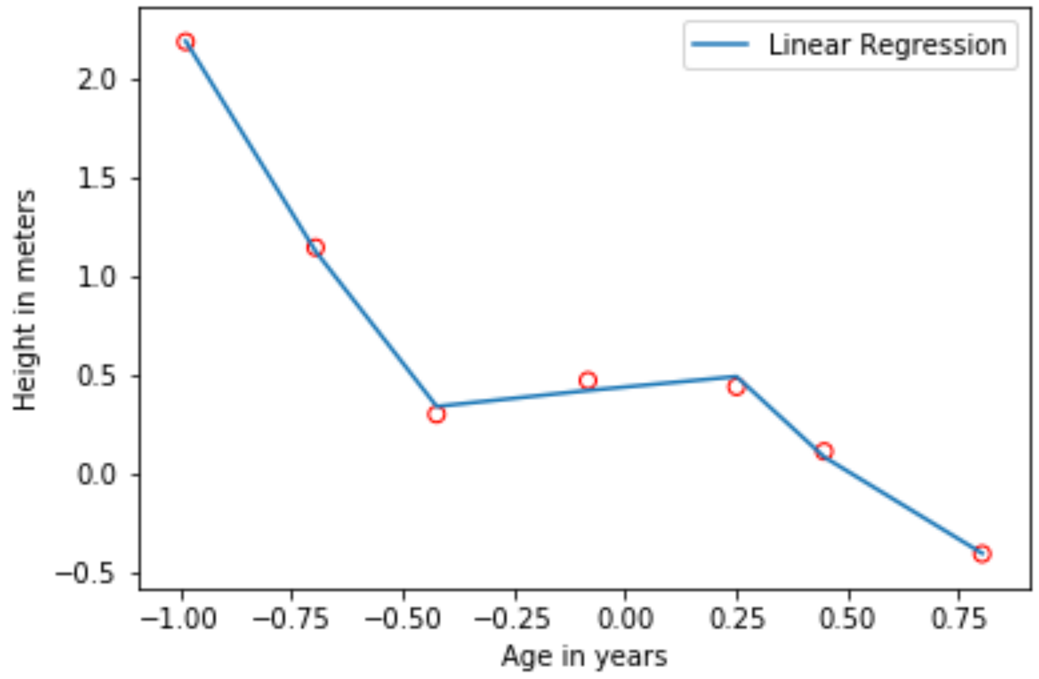
Plot of original data:

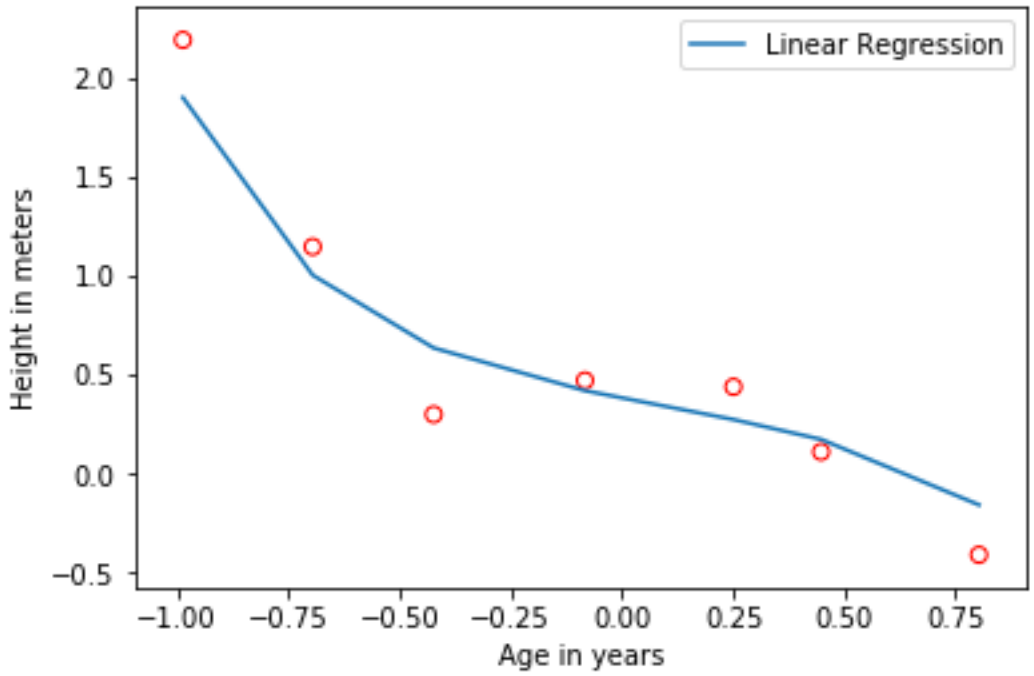


lambda: 0
theta: array([0.49917479, 0.83306615, -2.03845857, -6.93424088, 3.39424083,
5.75306609])
L2Norm: 9.8894299563366523

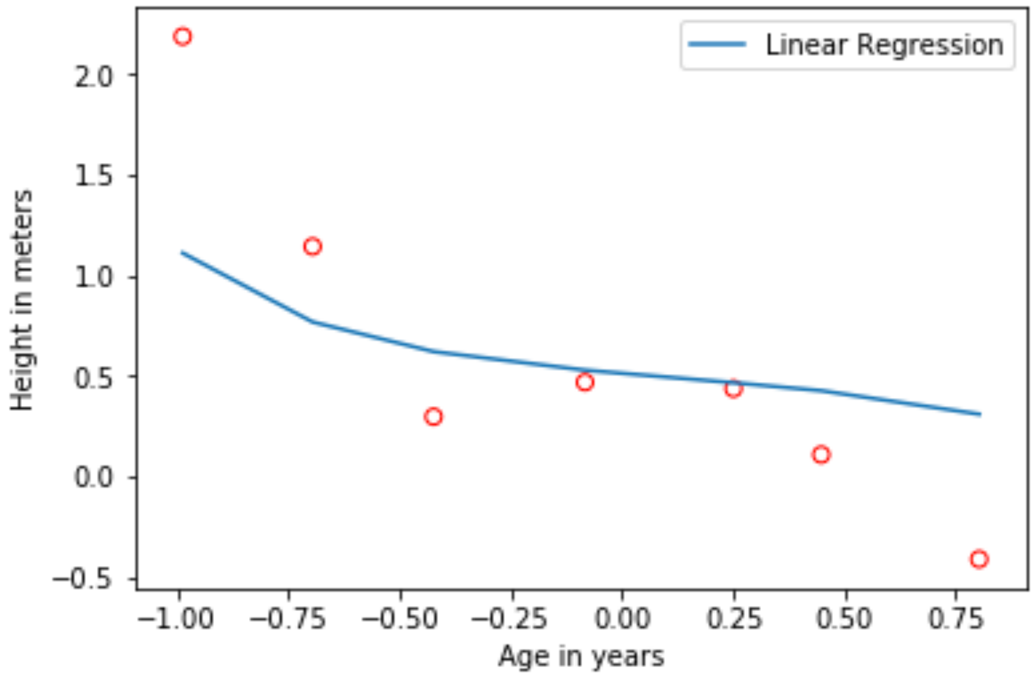


lambda: 1

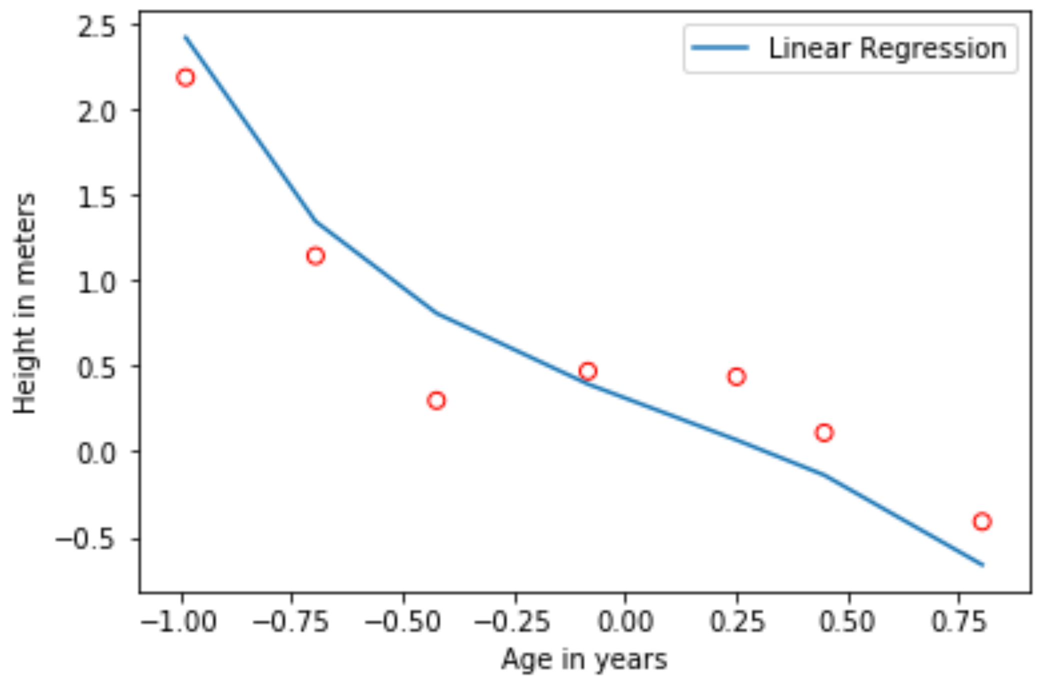
```
theta: array([ 0.38106374, -0.43781896,  0.12880159, -0.43609185,  0.1933533 ,
-0.37688109])
L2Norm: 0.85034452774484648
```



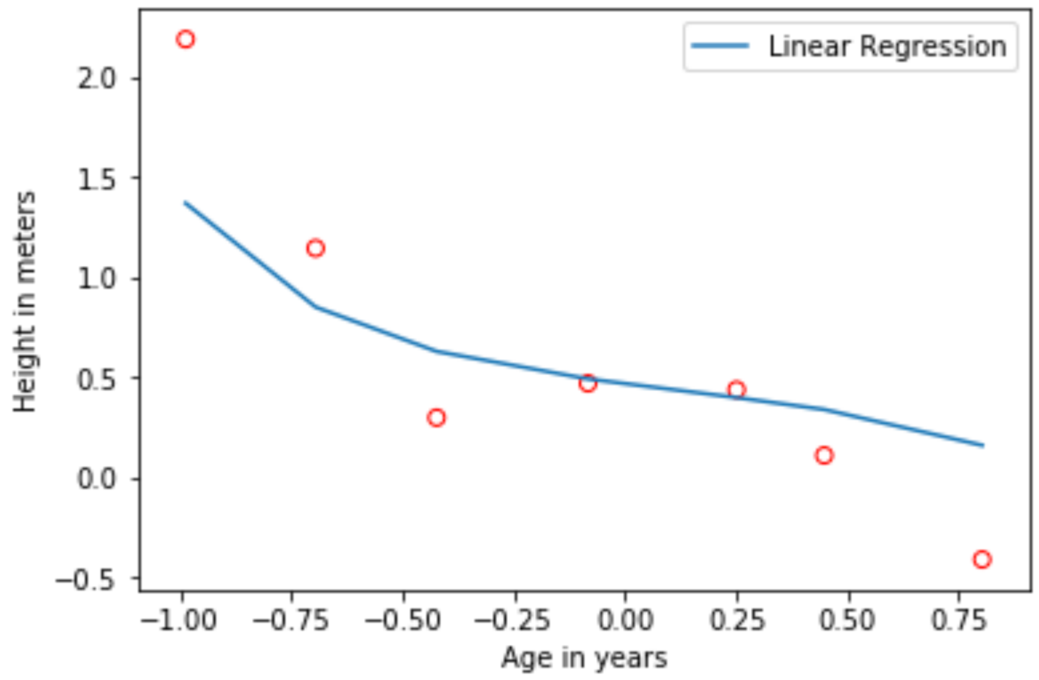
```
lambda: 10
theta: array([ 0.51357264, -0.19100471,  0.06437106, -0.15400044,  0.07903144,
-0.13137918])
L2Norm: 0.59296363841252353
```



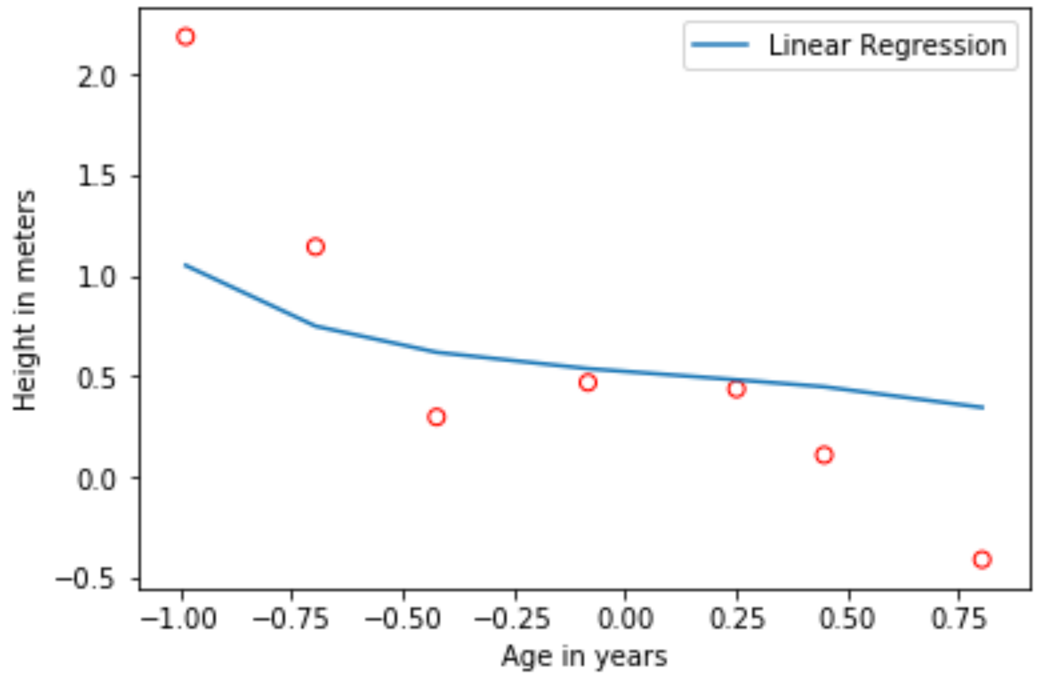
lambda: -1
theta: array([0.31089401, -0.98611001, 0.18116587, -0.48964457, 0.16984902,
-0.34777078])
L2Norm: 1.2212428600852612



lambda: 5 t
heta: array([0.46756365, -0.28432857, 0.09426909, -0.23548116, 0.1184942 ,
-0.20138849])
L2Norm: 0.64683439678612598



```
lambda: 12
theta: array([ 0.52443515, -0.16870046,  0.05701219, -0.13539704,  0.06972039,
-0.1154572 ])
L2Norm: 0.58588916953416492
```



Discuss how well you expect the theta to generalize to other data points in this distribution: I expect that the thetas that are closer to 0 will generalize better to other data points in this distribution. It matches to the current data points more closely, and I think will also match closer to points generated by the same distribution.

As lambda grows, the line that generalizes to the points gets less and less complex. It becomes almost linear as lambda goes to infinity. It draws a line that correlates most closely to the points with a degree of variability that relates to lambda. Lambda determines how far the line can be from the points it is correlating.

I think a lambda of a value that is very close to 0 will generalize to data the best. This could lead to over-training on our dataset, but I think a value of greater than 5 will not generalize well enough.

Code:

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 18 20:32:55 2017

@author: rditljtd
"""

import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from map_features import *

x = np.loadtxt('ax.dat')
y = np.loadtxt('ay.dat')

#Plot original data out
plt.scatter(x, y, facecolors='none', color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

#Get the number of examples
m = x.shape[0]
#Reshape x to be a 2D column vector
x.shape = (m,1)
#Add a column of ones to x
X = np.hstack([np.ones((m,1)), x, x**2, x**3, x**4, x**5])

theta = np.zeros(shape=(6,1))

lambdaa = [0, 1, 10, -1, 5, 12]

#L2norm = np.linalg.norm(X)

#print L2norm

for i in range(len(lambdaa)):

    matrix = np.diag([0, 1,1,1,1,1])
    #print matrix

    theta = np.linalg.inv(X.transpose().dot(X) + lambdaa[i]*(matrix)).dot(X.transpose().dot(y))
    print "lambda: " + `lambdaa[i]`
    print "theta: " + `theta`
    print "L2Norm: " + `np.linalg.norm(theta)`
    #Plot original data out
    plt.plot(X[:,1],np.dot(X,theta))
    plt.legend(['Linear Regression', 'Training Data'])
    plt.scatter(x, y, facecolors='none', color='red')
    plt.xlabel('Age in years')
    plt.ylabel('Height in meters')
    plt.show()
    i = i+1

#def computeCost(X, y, theta):
```

```
#    m=y.size
#    estimates = (X-y[:,None]).dot(theta).flatten()
#    squared_errors = (estimates) ** 2
#    sum_errors = (1 / (2 * m))*squared_errors.sum()
#    J = lambdaa*(theta)
#    return J
#
#    #sum_first = ((hypothesis(X)-y)**2).sum()
#    #sum_second = lambdaa*((theta**2).sum)
#    #J_theta = (1/(2*m))*(sum_first+sum_second)
#
#def gradientDescent(X, y, theta, lambdaa, iterations):
#    m = y.size
#    J_history = np.zeros(shape = (iterations, 1))
#
#    for i in range(iterations):
#        J_history[i,0] = computeCost(X, y, theta)
#
#    return theta, J_history
#
##theta, J_history = gradientDescent(X, y, theta, lambdaa, 10)
#
#print theta
```