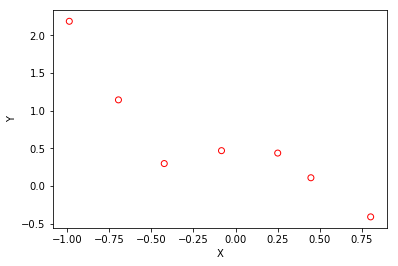
**Programming Assignment 2**

**Part A:**

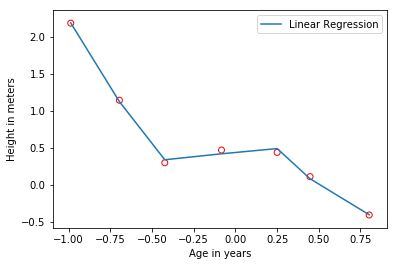
Plot of original data:



lambda: 0

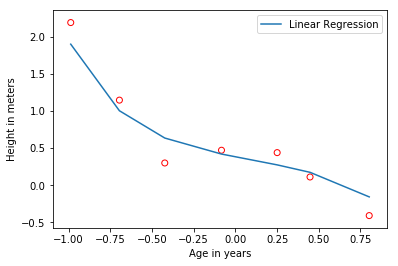
theta: array([ 0.49917479, 0.83306615, -2.03845857, -6.93424088, 3.39424083,
5.75306609])

L2Norm: 9.8894299563366523

lambda: 1

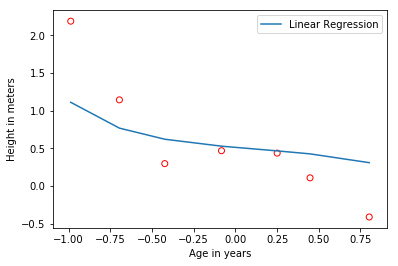
theta: array([ 0.38106374, -0.43781896, 0.12880159, -0.43609185, 0.1933533 ,
-0.37688109])

L2Norm: 0.85034452774484648

lambda: 10

theta: array([ 0.51357264, -0.19100471, 0.06437106, -0.15400044, 0.07903144,
-0.13137918])

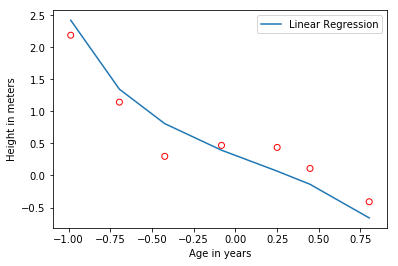
L2Norm: 0.59296363841252353



lambda: -1

theta: array([ 0.31089401, -0.98611001, 0.18116587, -0.48964457, 0.16984902,
-0.34777078])

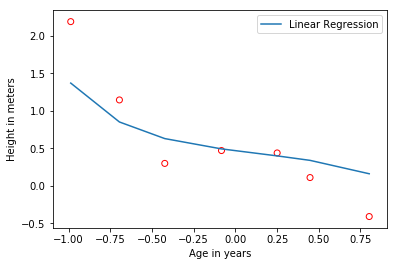
L2Norm: 1.2212428600852612



lambda: 5
t

heta: array([ 0.46756365, -0.28432857, 0.09426909, -0.23548116, 0.1184942 ,
-0.20138849])

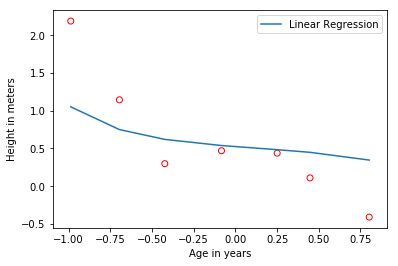
L2Norm: 0.64683439678612598



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:

1. #!/usr/bin/env python2
2. # -\*- coding: utf-8 -\*-
3. """
4. Created on Mon Sep 18 20:32:55 2017
6. @author: rditljtd
7. """
9. import numpy as np
10. from mpl\_toolkits.mplot3d import Axes3D
11. import matplotlib.pyplot as plt
12. from map\_features import \*
14. x = np.loadtxt('ax.dat')
15. y = np.loadtxt('ay.dat')
17. #Plot original data out
18. plt.scatter(x, y, facecolors='none', color='red')
19. plt.xlabel('X')
20. plt.ylabel('Y')
21. plt.show()
23. #Get the number of examples
24. m = x.shape[0]
25. #Reshape x to be a 2D column vector
26. x.shape = (m,1)
27. #Add a column of ones to x
28. X = np.hstack([np.ones((m,1)), x, x\*\*2, x\*\*3, x\*\*4, x\*\*5])
30. theta = np.zeros(shape=(6,1))
32. lambdaa = [0, 1, 10, -1, 5, 12]
34. #L2norm = np.linalg.norm(X)
36. #print L2norm
38. **for** i in range(len(lambdaa)):
40. matrix = np.diag([0, 1,1,1,1,1])
41. #print matrix
43. theta = np.linalg.inv(X.transpose().dot(X) + lambdaa[i]\*(matrix)).dot(X.transpose().dot(y))
44. print "lambda: " + `lambdaa[i]`
45. print "theta: " + `theta`
46. print "L2Norm: " + `np.linalg.norm(theta)`
47. #Plot original data out
48. plt.plot(X[:,1],np.dot(X,theta))
49. plt.legend(['Linear Regression', 'Training Data'])
50. plt.scatter(x, y, facecolors='none', color='red')
51. plt.xlabel('Age in years')
52. plt.ylabel('Height in meters')
53. plt.show()
54. i = i+1
56. #def computeCost(X, y, theta):
57. #    m=y.size
58. #    estimates = (X-y[:,None]).dot(theta).flatten()
59. #    squared\_errors = (estimates) \*\* 2
60. #    sum\_errors = (1 / (2 \* m))\*squared\_errors.sum()
61. #    J = lambdaa\*(theta)
62. #    return J
63. #
64. #    #sum\_first = ((hypothesis(X)-y)\*\*2).sum()
65. #    #sum\_second = lambdaa\*((theta\*\*2).sum)
66. #    #J\_theta = (1/(2\*m))\*(sum\_first+sum\_second)
67. #
68. #def gradientDescent(X, y, theta, lambdaa, iterations):
69. #    m = y.size
70. #    J\_history = np.zeros(shape = (iterations, 1))
71. #
72. #    for i in range(iterations):
73. #        J\_history[i,0] = computeCost(X, y, theta)
74. #
75. #    return theta, J\_history
76. #
77. ##theta, J\_history = gradientDescent(X, y, theta, lambdaa, 10)
78. #
79. #print theta