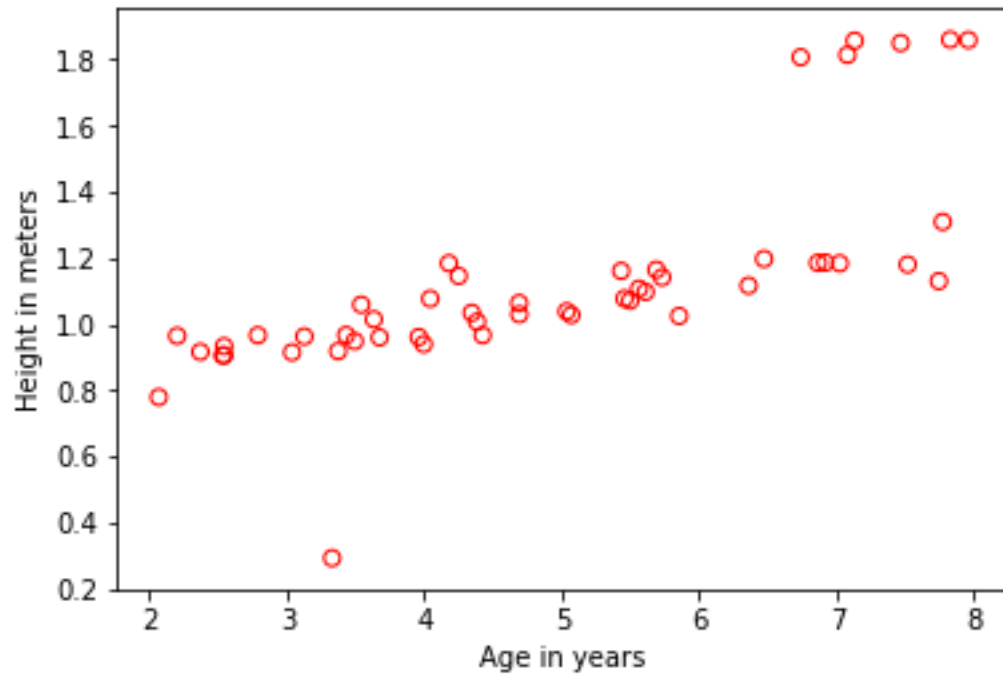


Programming Assignment 1

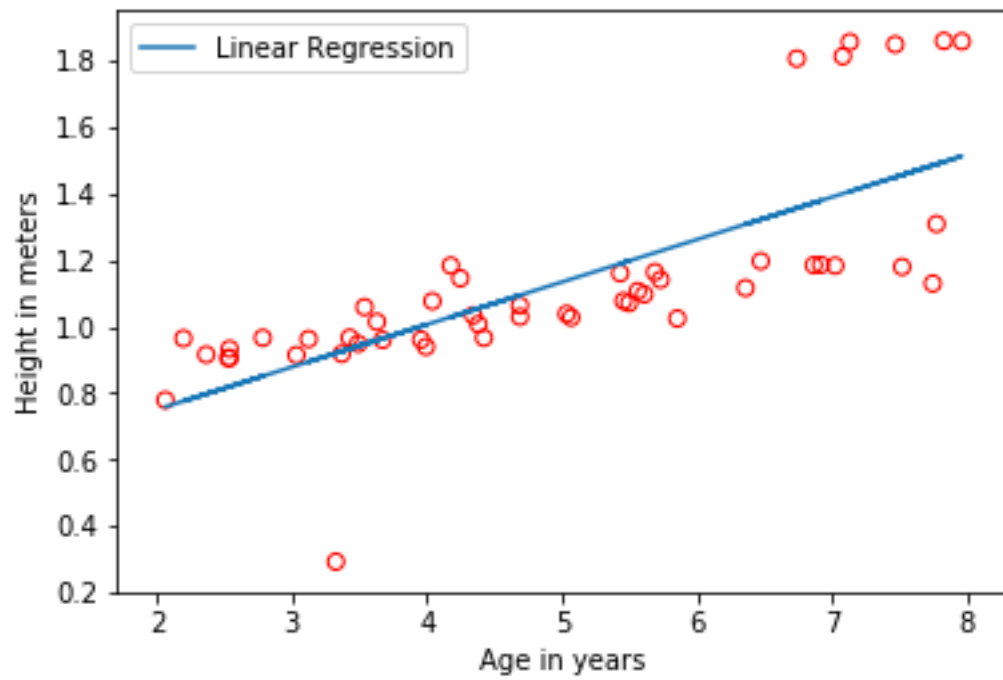
Part A

Raw data



Converged value of theta: $[[0.49421836] [0.12800497]]$

Plot of fit line

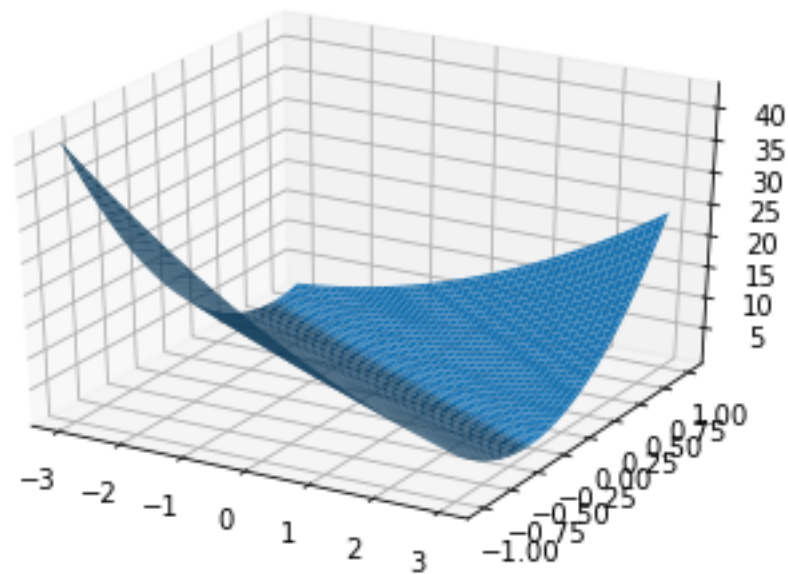


Prediction:

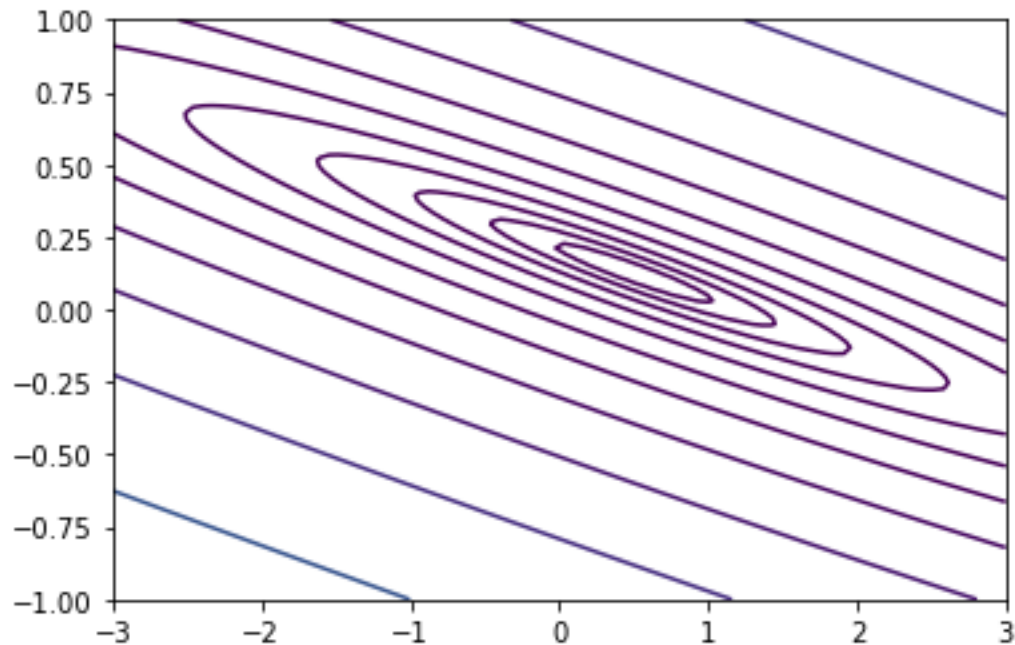
For age 3.5, we predict a height of 0.942236

For age 7, we predict a height of 1.390253

Plot of 3D surface of J:



Plot of contour of J:



What is the relationship between these plots (3D surface and contour) and the value of θ_0 and θ_1 that your implementation of gradient descent had found?

J_{theta} approaches a minimum at θ_0 and θ_1

Code:

```

1. # -*- coding: utf-8 -*-
2. """
3. Spyder Editor
4.
5. This is a temporary script file.
6. """
7.
8. import numpy as np
9. from mpl_toolkits.mplot3d import Axes3D
10. import matplotlib.pyplot as plt
11.
12. x = np.loadtxt('ax.dat')
13. y = np.loadtxt('ay.dat')
14.
15. #Plot original data out
16. plt.scatter(x, y, facecolors='none', color='red')
17. plt.xlabel('Age in years')
18. plt.ylabel('Height in meters')
19. plt.show()
20.
21. #Get the number of examples
22. m = x.shape[0]
23. #Reshape x to be a 2D column vector
24. x.shape = (m,1)
25. #Add a column of ones to x
26. X = np.hstack([np.ones((m,1)), x])
27.
28.

```

```

29. #initialize theta
30. theta = np.zeros(shape=(2,1))
31.
32.
33. #gradient descent
34. alpha = .07
35. loops = 1300
36.
37. def computeCost(X, y, theta):
38.     m = y.size
39.     estimates = X.dot(theta).flatten()
40.     squaredErrors = (estimates - y) ** 2
41.     J = (1.0 / (2 * m)) * squaredErrors.sum()
42.     return J
43.
44. def gradientDescent(X, y, theta, alpha, iterations):
45.     m=y.size
46.     J_history = np.zeros(shape = (iterations, 1))
47.
48.     for i in range(iterations):
49.         estimates = X.dot(theta).flatten()
50.         err_x1 = (estimates - y) * X[:, 0]
51.         err_x2 = (estimates - y) * X[:, 1]
52.
53.         theta[0][0] = theta[0][0] - alpha * (1.0/m) * err_x1.sum()
54.         theta[1][0] = theta[1][0] - alpha * (1.0/m) * err_x2.sum()
55.
56.         J_history[i,0] = computeCost(X, y, theta)
57.
58.     return theta, J_history
59.
60.
61. theta, J_history = gradientDescent(X, y, theta, alpha, 1300)
62.
63. print theta
64.
65. predict1 = np.array([1, 3.5]).dot(theta).flatten()
66. print 'For age 3.5, we predict a height of %f' % (predict1)
67. predict2 = np.array([1, 7]).dot(theta).flatten()
68. print 'For age 7, we predict a height of %f' % (predict2)
69.
70.
71. plt.plot(X[:,1],np.dot(X,theta))
72. plt.legend(['Linear Regression', 'Training Data'])
73. plt.scatter(x, y, facecolors='none', color='red')
74. plt.xlabel('Age in years')
75. plt.ylabel('Height in meters')
76. plt.show()
77.
78.
79.
80. #Display Surface Plot of J
81. t0 = np.linspace(-3,3,100)
82. t1 = np.linspace(-1,1,100)
83. t0.shape = (len(t0),1)
84. t1.shape = (len(t1),1)
85. T0, T1 = np.meshgrid(t0,t1)
86. J_vals = np.zeros((len(t0),len(t1)))
87. for i in range(len(t0)):
88.     for j in range(len(t1)):
89.         t = np.hstack([t0[i], t1[j]])

```

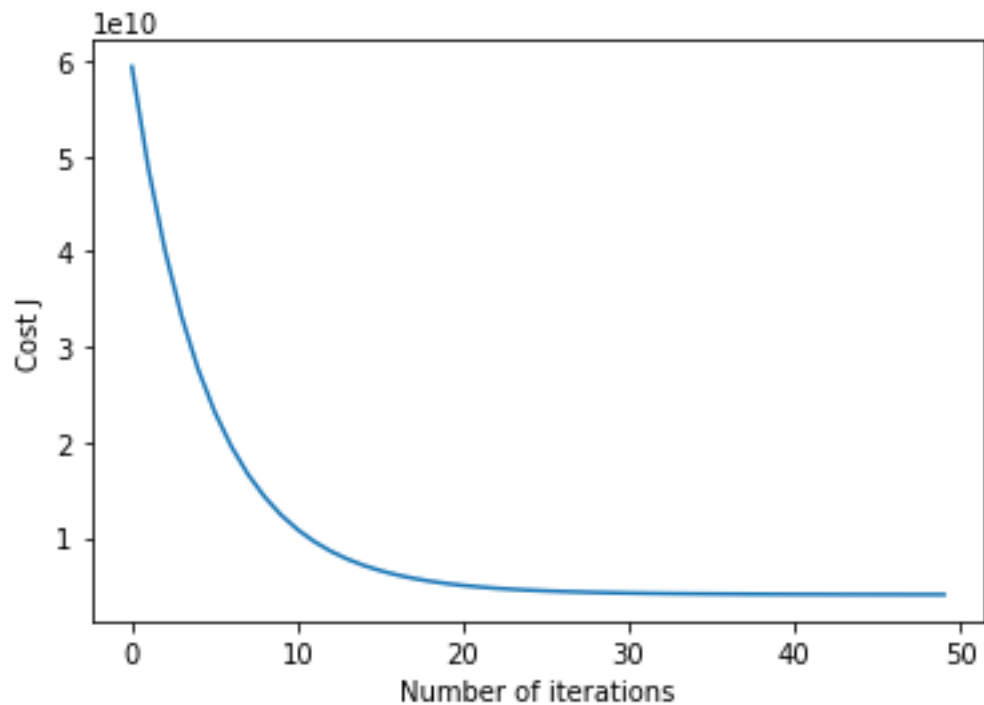
```

90.         J_vals[i,j] = computeCost(X, y, t)
91.
92. #Because of the way meshgrids work with plotting surfaces
93. #we need to transpose J to show it correctly
94. J_vals = J_vals.T
95. fig = plt.figure()
96. ax = fig.gca(projection='3d')
97. ax.plot_surface(T0,T1,J_vals)
98. plt.show()
99. plt.close()
100.     #Display Contour Plot of J
101.     5
102.     plt.contour(T0,T1,J_vals, np.logspace(-2,2,15))
103.     plt.show()

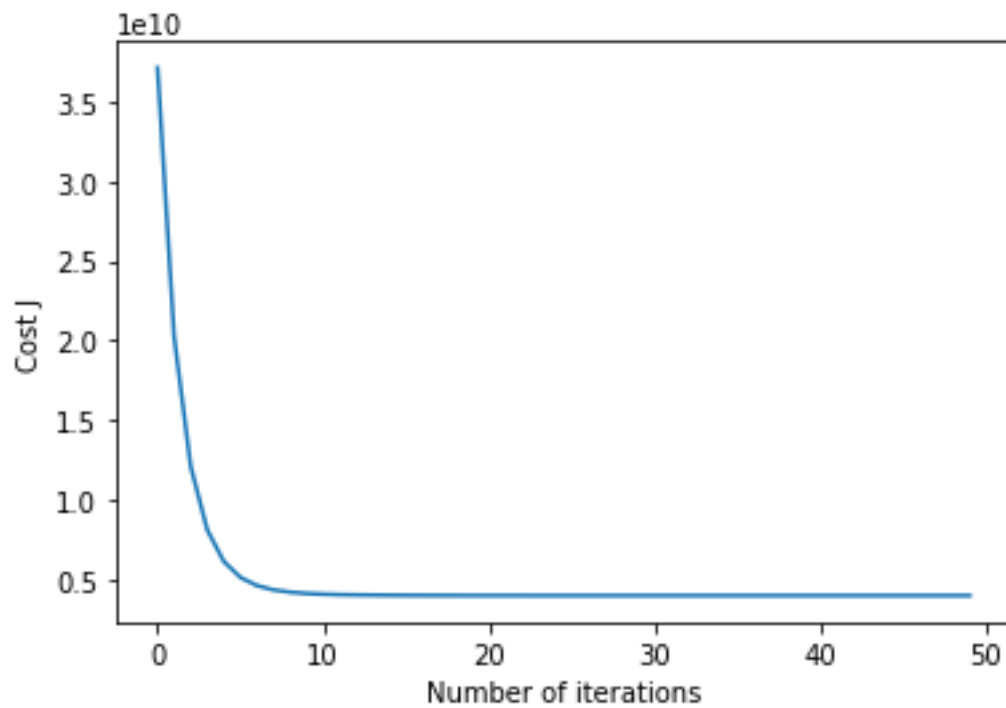
```

Part B

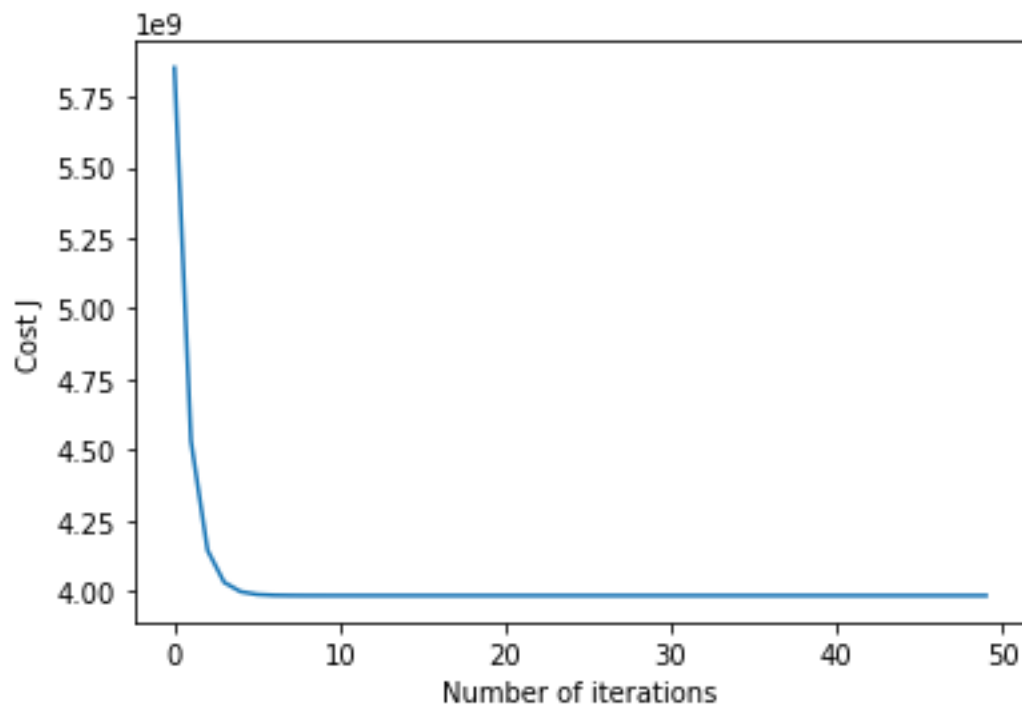
Learning rate of .1



Learning rate of .3



Learning rate of 1.0



Converged value of theta for 1.0 learning rate: $\begin{bmatrix} 353178.61702128 \\ 114973.94461206 \\ -3702.28591238 \end{bmatrix}$

Prediction of price of house with 3 bedrooms that is 1650 sqft: 303613.809912

Code:

```
1. #!/usr/bin/env python2
2. # -*- coding: utf-8 -*-
3. """
4. Created on Mon Sep 11 20:23:00 2017
5.
6. @author: rditljtd
7. """
8.
9. import numpy as np
10. from mpl_toolkits.mplot3d import Axes3D
11. import matplotlib.pyplot as plt
12.
13. x = np.loadtxt('bx.dat')
14. y = np.loadtxt('by.dat')
15.
16.
17. test = [1650, 3]
18.
19. #Get the number of examples
20. m = x.shape[0]
21.
22. #Reshape x to be a 2D column vector
23. x.shape = (m,2)
24.
25. sigma = np.std(x,axis=0) #std
26. mu = np.mean(x,axis=0) #mean
27. x = (x-mu) / sigma #adjustment
28.
29. test = (test-mu) / sigma
30.
31. #Add a column of ones to x
32. X = np.hstack([np.ones((m,1)), x])
33.
34. #initialize theta
35. theta = np.zeros(shape=(3,1)) #Initialize theta
36. #print theta
37. alpha = 1.#Your learning rate#
38. #J = []
39. iterations = 50
40. J_history = np.zeros(shape = (iterations, 1))
41.
42. #Closed-form solution
43.
44. theta2 = np.linalg.inv(X.transpose().dot(X)).dot(X.transpose().dot(y))
45. print theta2
46.
47. predict2 = np.array([1, test[0], test[1]]).dot(theta2).flatten()
48. print 'For a house with 3 bedrooms and 1650 sqft, we predict a price of %f' % (p
redict2)
49.
50.
51. #gradient descent solution
52.
53. def computeCost(X, y, theta):
54.     m = y.size
55.     estimates = X.dot(theta).flatten()
56.     squaredErrors = (estimates - y) ** 2
```

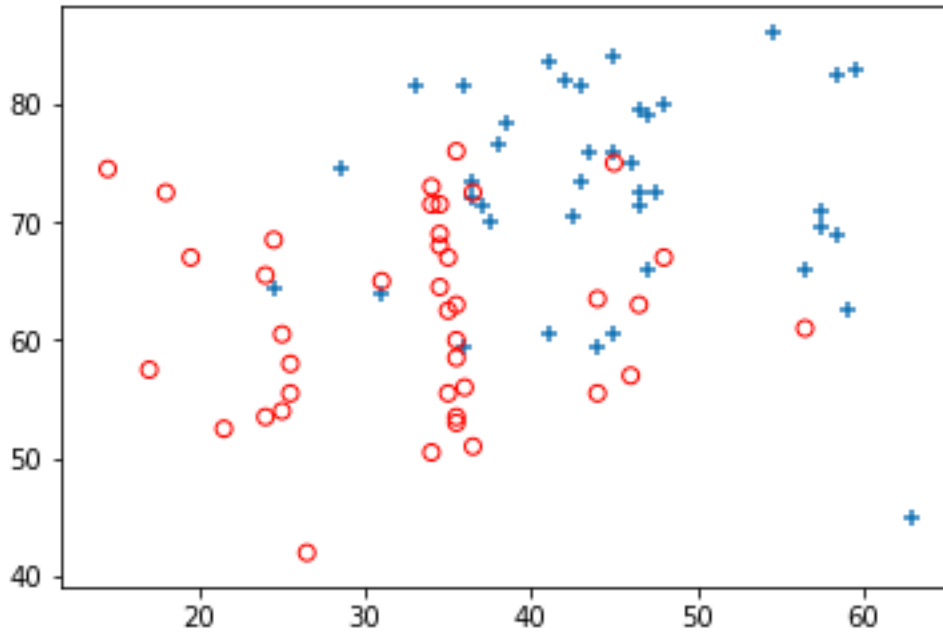
```

57.     J = (1.0 / (2 * m)) * squaredErrors.sum()
58.     return J
59.
60. def gradientDescent(X, y, theta, alpha, iterations):
61.     m=y.size
62.
63.     estimates = X.dot(theta).flatten()
64.     err_x1 = (estimates - y) * X[:, 0]
65.     err_x2 = (estimates - y) * X[:, 1]
66.     err_x3 = (estimates - y) * X[:, 2]
67.     #print "err_x1: " + `err_x1` + " err_x2: " + `err_x2` + " err_x3: " + `err_x3`
68.
69.     theta[0][0] = theta[0][0] - alpha * (1.0/m) * err_x1.sum()
70.     theta[1][0] = theta[1][0] - alpha * (1.0/m) * err_x2.sum()
71.     theta[2][0] = theta[2][0] - alpha * (1.0/m) * err_x3.sum()
72.
73.
74.
75.     return theta
76.
77. for i in range(iterations):
78.     theta = gradientDescent(X, y, theta, alpha, iterations)#might need to take i
    out
79.     J_history[i] = computeCost(X, y, theta)
80.
81. #Now plot J
82. plt.plot(range(iterations), J_history)
83. plt.xlabel('Number of iterations')
84. plt.ylabel('Cost J')
85.
86. print theta
87.
88. predict1 = np.array([1, test[0], test[1]]).dot(theta).flatten()
89. print 'For a house with 3 bedrooms and 1650 sqft, we predict a price of %f' % (p
    redict1)

```

Part C

Plot of raw data:



I spent a good 4 hours on Part C last night and could not figure out how Newton's method worked. I could not get the math programmed correctly. I tried tweaking it for hours, and nothing I did seemed to work.

Here is the code I had when I went to sleep last night.

Code:

```
1.  #!/usr/bin/env python2
2.  # -*- coding: utf-8 -*-
3.  """
4.  Created on Mon Sep 11 20:23:00 2017
5.
6.  @author: rditljtd
7.  """
8.
9.  import numpy as np
10. from mpl_toolkits.mplot3d import Axes3D
11. import matplotlib.pyplot as plt
12. from scipy.special import expit
13.
14. x = np.loadtxt('cx.dat')
15. y = np.loadtxt('cy.dat')
16.
17.
18. #Get the number of examples
19. m = x.shape[0]
20.
21. #Reshape x to be a 2D column vector
22. x.shape = (m,2)
23.
24. #Add a column of ones to x
25. X = np.hstack([np.ones((m,1)), x])
```

```

26.
27. #initialize theta
28. theta = np.zeros(shape=(3,1)) #Initialize theta
29. #print theta
30. #alpha = .3#Your learning rate#
31. #J = []
32. iterations = 15
33. J_history = np.zeros(shape = (iterations, 1))
34.
35. pos = np.nonzero(y)
36. neg = np.where(y==0)[0]
37.
38. plt.scatter(x[pos,0],x[pos,1],marker='+')
39. plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o', color='r')
40. plt.show()
41.
42.
43. #Newtons method solution
44.
45. def sigmoid(z):
46.     print "z: " + `z`
47.     toreturn = expit(z)
48.     print "sigmoid: " + `toreturn`
49.     return toreturn
50.
51. def hypothesis(X, theta):
52.     print "here: " + `theta`
53.     print "there: " + `X`
54.     toreturn = (sigmoid(np.transpose(theta))*(X))
55.     print "hypothesis: " + `toreturn`
56.     return toreturn
57.
58.
59. def computeCost(X, y, theta):
60.     m = y.size
61.     toreturn = ((1.0/m) * (-y.dot(np.log(hypothesis(X, theta))) - (1-
y).dot(np.log(1-hypothesis(X, theta)))).sum())
62.     print "computeCost: " + `toreturn`
63.     return toreturn
64.
65. def gradientDescent(X, y, theta):
66.     m = y.size
67.     minus = hypothesis(X, theta).subtract(hypothesis(X,theta), y)
68.     toreturn = (1.0/m)*(minus.dot(X))
69.     print "gradientDescent: " + `toreturn`
70.     return toreturn
71.
72. def hessian(X, y, theta):
73.     m = y.size
74.     minus = 1-hypothesis(X,theta)*(hypothesis(X, theta))
75.     xTrans = (X.transpose()).dot(X)
76.     result = minus.dot(xTrans)
77.     print np.sum(result)
78.     toreturn = (1.0/m)*((minus).dot(xTrans)).sum()
79.     print "hessian: " + `toreturn`
80.     return toreturn
81.
82. def newtonsMethod(X, y, theta, iterations):
83.     m=y.size
84.
85.     for i in range(iterations):

```

```
86.         J_history[i] = computeCost(X, y, theta)
87.         theta = (theta -
(1/(hessian(X, y, theta))) * gradientDescent(X, y, theta))
88.
89.     print theta, J_history
90.     return theta, J_history
91.
92. theta, J_history = newtonsMethod(X, y, theta, 15)
93.
```