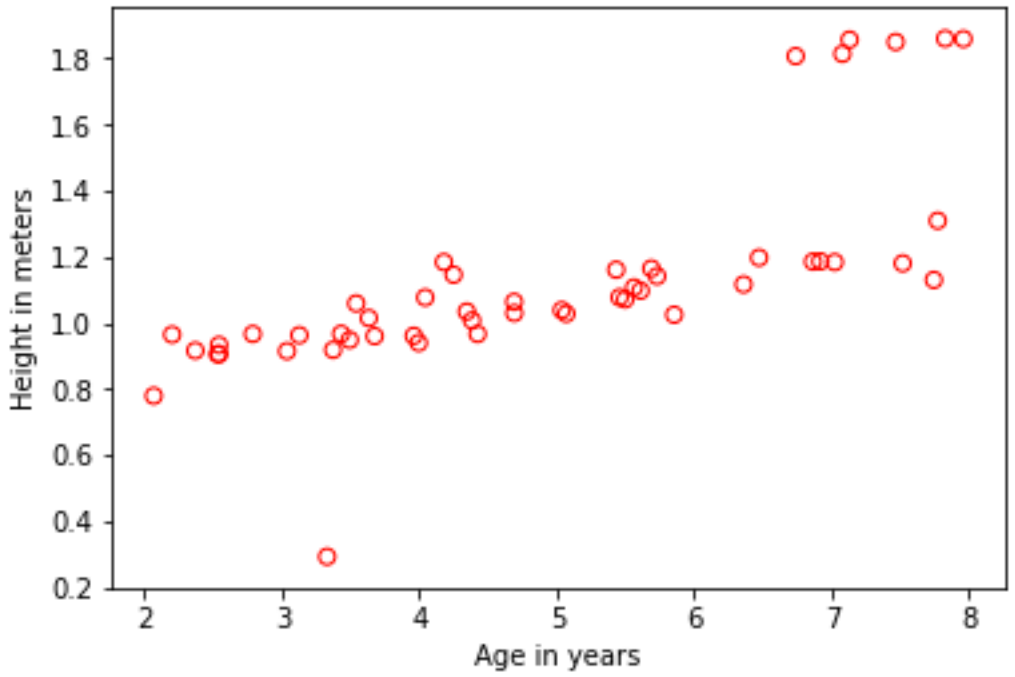
Programming Assignment 1

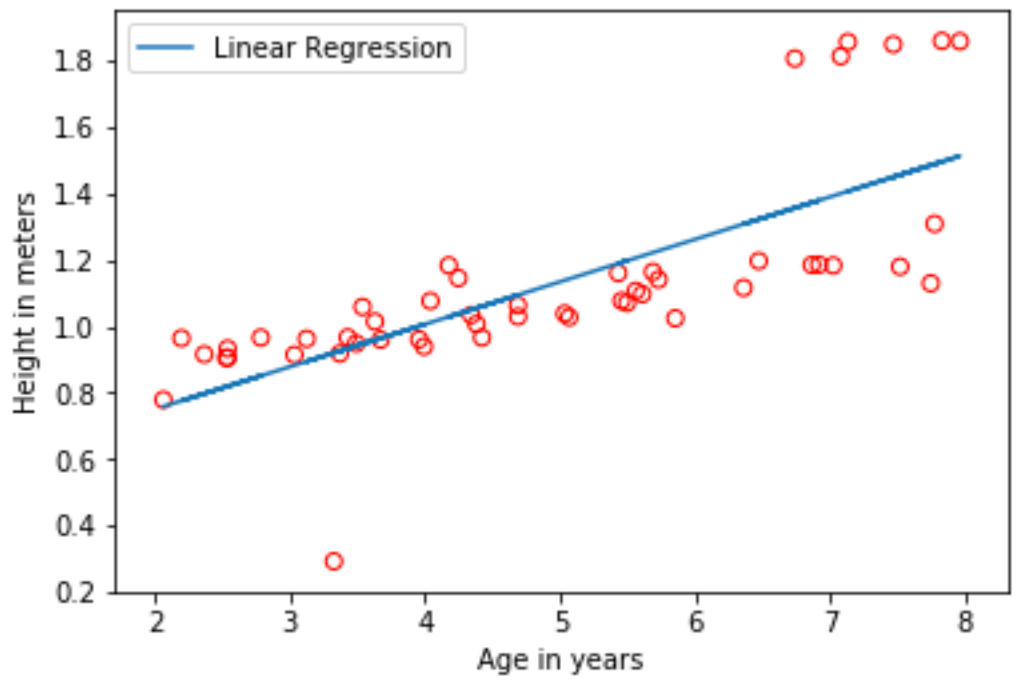
Part A

Raw data



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

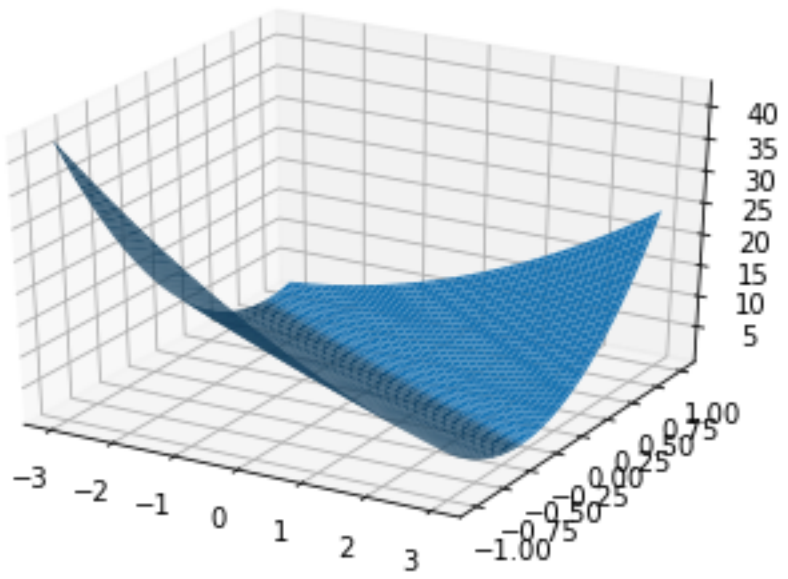
Plot of fit line



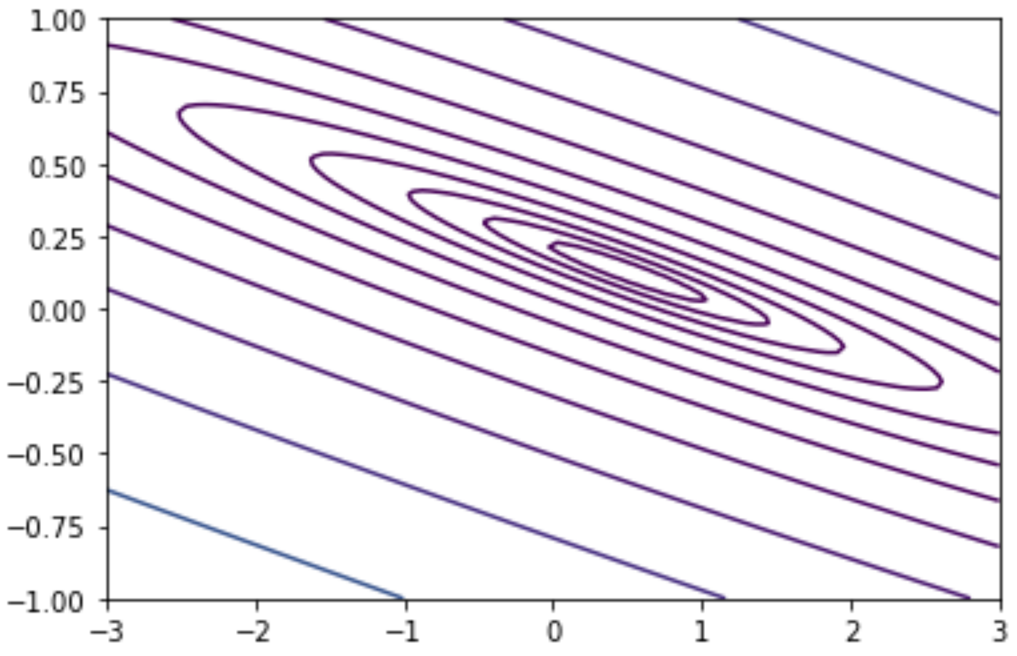
Prediction:

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 θ0 and θ1 that your implementation of gradient descent had found?

J_theta approaches a minimum at theta_0 and theta_1

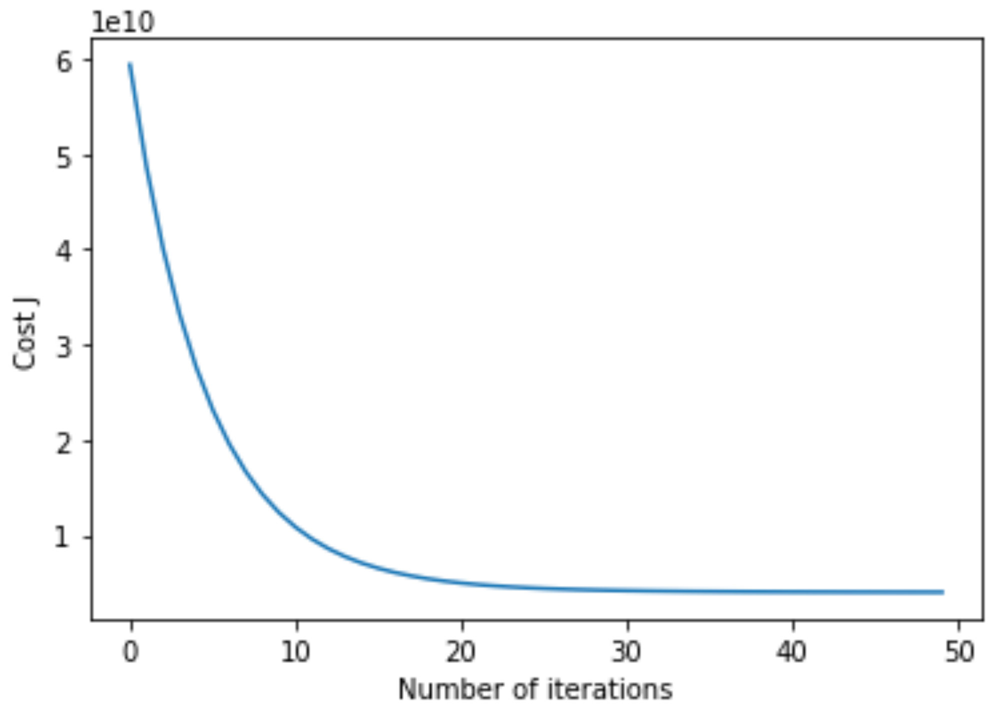
Code:

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

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

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

#Plot original data out
plt.scatter(x, y, facecolors='none', color='red')
plt.xlabel('Age in years')
plt.ylabel('Height in meters')
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])


```

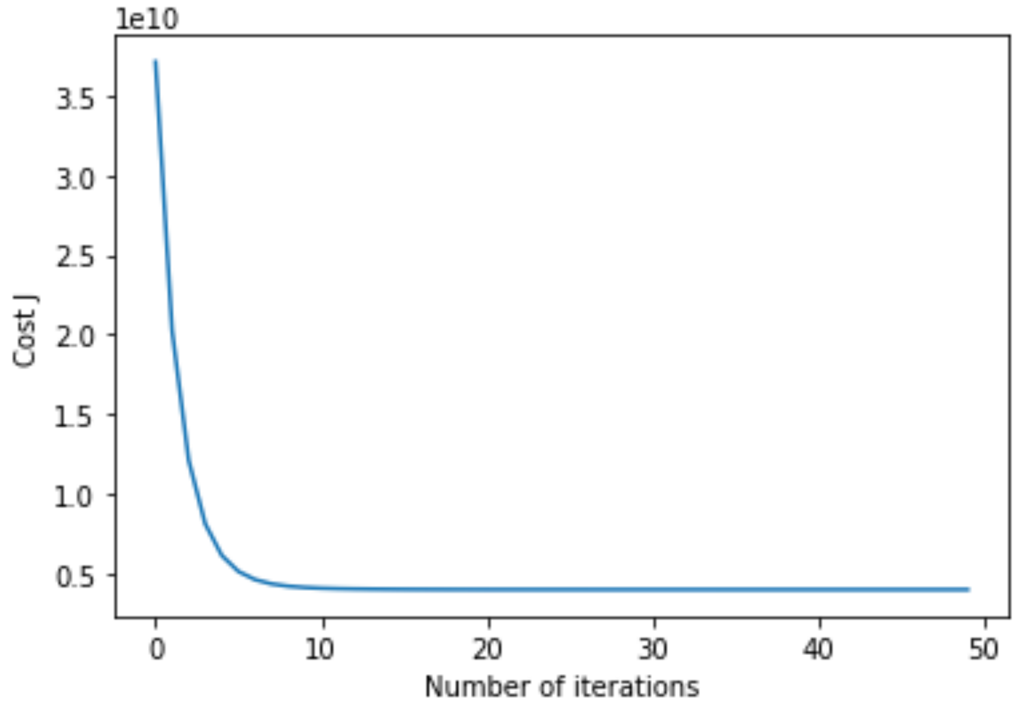
```python
#initialize theta
theta = np.zeros(shape=(2,1))


#gradient descent
alpha = .07
loops = 1300

def computeCost(X, y, theta):
    m = y.size
    estimates = X.dot(theta).flatten()
    squaredErrors = (estimates - y) ** 2
    J = (1.0 / (2 * m)) * squaredErrors.sum()
    return J

def gradientDescent(X, y, theta, alpha, iterations):
    m=y.size
    J_history = np.zeros(shape = (iterations, 1))

    for i in range(iterations):
        estimates = X.dot(theta).flatten()
        err_x1 = (estimates - y) * X[:, 0]
        err_x2 = (estimates - y) * X[:, 1]

        theta[0][0] = theta[0][0] - alpha * (1.0/m) * err_x1.sum()
        theta[1][0] = theta[1][0] - alpha * (1.0/m) * err_x2.sum()

        J_history[i,0] = computeCost(X, y, theta)

    return theta, J_history


theta, J_history = gradientDescent(X, y, theta, alpha, 1300)

print theta

predict1 = np.array([1, 3.5]).dot(theta).flatten()
print 'For age 3.5, we predict a height of %f' % (predict1)
predict2 = np.array([1, 7]).dot(theta).flatten()
print 'For age 7, we predict a height of %f' % (predict2)


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()



#Display Surface Plot of J
t0 = np.linspace(-3,3,100)
t1 = np.linspace(-1,1,100)
t0.shape = (len(t0),1)
t1.shape = (len(t1),1)
T0, T1 = np.meshgrid(t0,t1)
J_vals = np.zeros((len(t0),len(t1)))
for i in range(len(t0)):
    for j in range(len(t1)):
        t = np.hstack([t0[i], t1[j]])
        J_vals[i,j] = computeCost(X, y, t)

#Because of the way meshgrids work with plotting surfaces
#we need to transpose J to show it correctly
J_vals = J_vals.T
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(T0,T1,J_vals)
plt.show()
plt.close()
       #Display Contour Plot of J
       5
       plt.contour(T0,T1,J_vals, np.logspace(-2,2,15))
       plt.show()
```

Part B

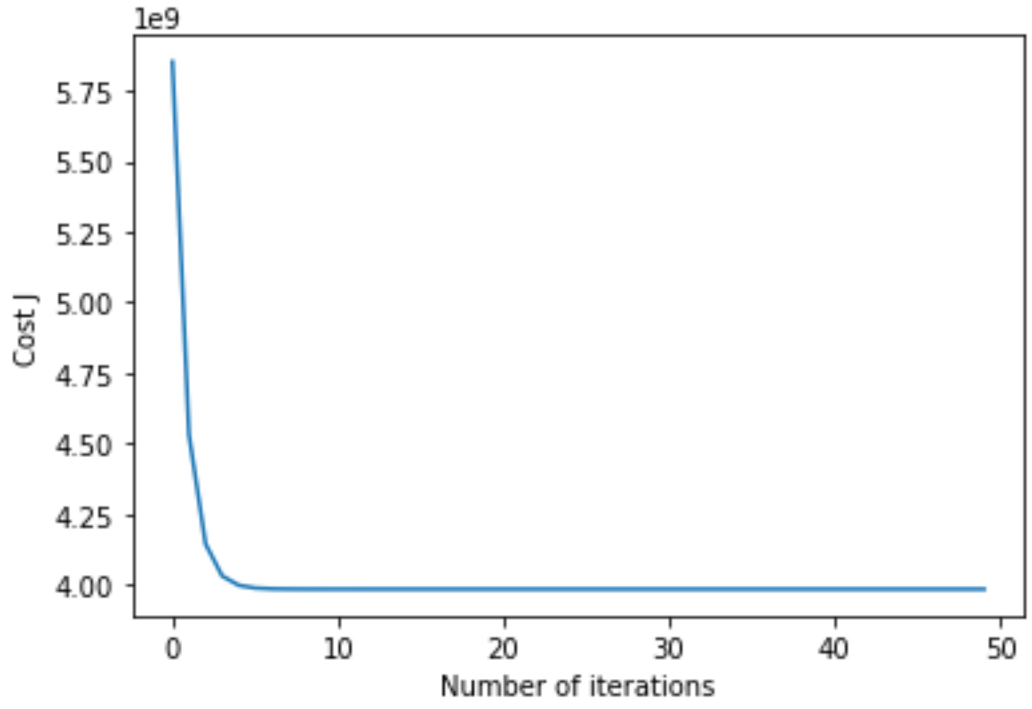
Learning rate of .1



Learning rate of .3



Learning rate of 1.0



Converged value of theta for 1.0 learning rate: [[353178.61702128]
[114973.94461206] [-3702.28591238]]

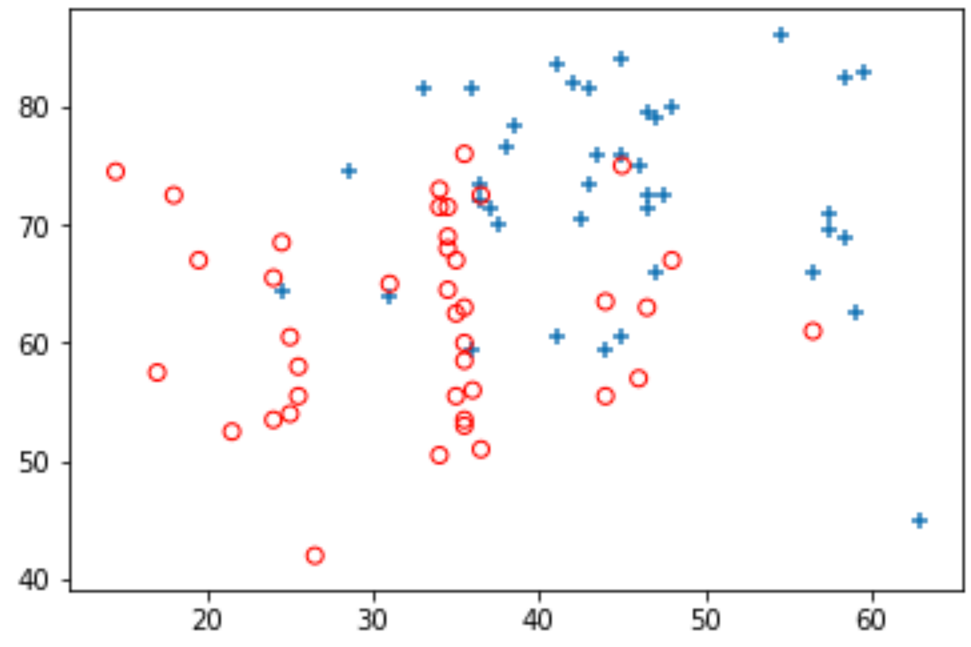
Prediction of price of house with 3 bedrooms that is 1650 sqft: 303613.809912

Code:

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 11 20:23:00 2017

@author: rditljtd
"""

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

x = np.loadtxt('bx.dat')
y = np.loadtxt('by.dat')


test = [1650, 3]

#Get the number of examples
m = x.shape[0]

#Reshape x to be a 2D column vector
x.shape = (m,2)

sigma = np.std(x,axis=0) #std
mu = np.mean(x,axis=0) #mean
x = (x-mu) / sigma #adjustment

test = (test-mu) / sigma

#Add a column of ones to x
X = np.hstack([np.ones((m,1)), x])

#initialize theta
theta = np.zeros(shape=(3,1)) #Initialize theta
#print theta
alpha = 1.#Your learning rate#
#J = []
iterations = 50
J_history = np.zeros(shape = (iterations, 1))

#Closed-form solution

theta2 = np.linalg.inv(X.transpose().dot(X)).dot(X.transpose().dot(y))
print theta2

predict2 = np.array([1, test[0], test[1]]).dot(theta2).flatten()
print 'For a house with 3 bedrooms and 1650 sqft, we predict a price of %f' % (p
redict2)


#gradient descent solution

def computeCost(X, y, theta):
    m = y.size
    estimates = X.dot(theta).flatten()
    squaredErrors = (estimates - y) ** 2
    J = (1.0 / (2 * m)) * squaredErrors.sum()
    return J

def gradientDescent(X, y, theta, alpha, iterations):
    m=y.size

    estimates = X.dot(theta).flatten()
    err_x1 = (estimates - y) * X[:, 0]
    err_x2 = (estimates - y) * X[:, 1]
    err_x3 = (estimates - y) * X[:, 2]
        #print "err_x1: " + `err_x1` + " err_x2: " + `err_x2` + " err_x3: " + `err_x3`

    theta[0][0] = theta[0][0] - alpha * (1.0/m) * err_x1.sum()
    theta[1][0] = theta[1][0] - alpha * (1.0/m) * err_x2.sum()
    theta[2][0] = theta[2][0] - alpha * (1.0/m) * err_x3.sum()



    return theta

for i in range(iterations):
    theta = gradientDescent(X, y, theta, alpha, iterations)#might need to take i out
    J_history[i] = computeCost(X, y, theta)

#Now plot J
plt.plot(range(iterations), J_history)
plt.xlabel('Number of iterations')
plt.ylabel('Cost J')

print theta

predict1 = np.array([1, test[0], test[1]]).dot(theta).flatten()
print 'For a house with 3 bedrooms and 1650 sqft, we predict a price of %f' % (predict1)
```

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:

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 11 20:23:00 2017

@author: rditljtd
"""

import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from scipy.special import expit

x = np.loadtxt('cx.dat')
y = np.loadtxt('cy.dat')


#Get the number of examples
m = x.shape[0]

#Reshape x to be a 2D column vector
x.shape = (m,2)

#Add a column of ones to x
X = np.hstack([np.ones((m,1)), x])
```

```python

#initialize theta
theta = np.zeros(shape=(3,1)) #Initialize theta
#print theta
#alpha = .3#Your learning rate#
#J = []
iterations = 15
J_history = np.zeros(shape = (iterations, 1))

pos = np.nonzero(y)
neg = np.where(y==0)[0]

plt.scatter(x[pos,0],x[pos,1],marker='+')
plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o', color='r')
plt.show()


#Newtons method solution

def sigmoid(z):
    print "z: " + `z`
    toreturn = expit(z)
    print "sigmoid: " + `toreturn`
    return toreturn

def hypothesis(X, theta):
    print "here: " + `theta`
    print "there: " + `X`
    toreturn = (sigmoid(np.transpose(theta))*(X))
    print "hypothesis: " + `toreturn`
    return toreturn


def computeCost(X, y, theta):
    m = y.size
    toreturn = ((1.0/m) * (-y.dot(np.log(hypothesis(X, theta))) - (1-
y).dot(np.log(1-hypothesis(X, theta)))).sum())
    print "computeCost: " + `toreturn`
    return toreturn

def gradientDescent(X, y, theta):
    m = y.size
    minus = hypothesis(X, theta).subtract(hypothesis(X,theta), y)
    toreturn = (1.0/m)*(minus.dot(X))
    print "gradientDescent: " + `toreturn`
    return toreturn

def hessian(X, y, theta):
    m = y.size
    minus = 1-hypothesis(X,theta)*(hypothesis(X, theta))
    xTrans = (X.transpose()).dot(X)
    result = minus.dot(xTrans)
    print np.sum(result)
    toreturn = (1.0/m)*((minus).dot(xTrans)).sum()
    print "hessian: " + `toreturn`
    return toreturn

def newtonsMethod(X, y, theta, iterations):
    m=y.size

    for i in range(iterations):
        J_history[i] = computeCost(X, y, theta)
        theta = (theta - (1/(hessian(X, y, theta))) * gradientDescent(X, y, theta))

    print theta, J_history
    return theta, J_history

theta, J_history = newtonsMethod(X, y, theta, 15)

```