In [1]:
```
from numpy import *
num_movies=10
num_users=5
ratings=random.randint(11,size=(num_movies,num_users))
print ratings
```

```
[[10  1  0  1  2]
 [ 3  9  9  3  4]
 [ 8  2  5  2 10]
 [ 7  5  6  0  9]
 [ 2  7  9  7  5]
 [ 3  5  3  4  2]
 [ 6  5  0  0  1]
 [ 2  3  2  4  2]
 [ 9  9  1  7  0]
 [ 4  0 10  6  1]]
```

In [2]:
```
didrate =(ratings!=0)*1
print didrate
```

```
[[1 1 0 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 0 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 0 0 1]
 [1 1 1 1 1]
 [1 1 1 1 0]
 [1 0 1 1 1]]
```

In [3]:
```
rajeev_ratings= zeros((num_movies,1))
print rajeev_ratings
```

```
[[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
```

In [4]:
```
rajeev_ratings[6]=9
rajeev_ratings[5]=7
rajeev_ratings[2]=4
rajeev_ratings[9]=3
print rajeev_ratings
```

```
[[ 0.]
 [ 0.]
 [ 4.]
 [ 0.]
 [ 0.]
 [ 7.]
 [ 9.]
 [ 0.]
 [ 0.]
 [ 3.]]
```

In [5]:
```
ratings=append(rajeev_ratings,ratings,axis=1)
print ratings
```

```
[[  0.  10.   1.   0.   1.   2.]
 [  0.   3.   9.   9.   3.   4.]
 [  4.   8.   2.   5.   2.  10.]
 [  0.   7.   5.   6.   0.   9.]
 [  0.   2.   7.   9.   7.   5.]
 [  7.   3.   5.   3.   4.   2.]
 [  9.   6.   5.   0.   0.   1.]
 [  0.   2.   3.   2.   4.   2.]
 [  0.   9.   9.   1.   7.   0.]
 [  3.   4.   0.  10.   6.   1.]]
```

In [6]:
```
didrate=append(((rajeev_ratings!=0)*1),didrate,axis=1)
print didrate
```

```
[[0 1 1 0 1 1]
 [0 1 1 1 1 1]
 [1 1 1 1 1 1]
 [0 1 1 1 0 1]
 [0 1 1 1 1 1]
 [1 1 1 1 1 1]
 [1 1 1 0 0 1]
 [0 1 1 1 1 1]
 [0 1 1 1 1 0]
 [1 1 0 1 1 1]]
```

In [7]:
```
a=[10,20,30]
avg=mean(a)
print avg
```

```
20.0
```

In [8]:
```
a=[10-avg,20-avg,30-avg]
print a
```

```
[-10.0, 0.0, 10.0]
```

In [9]:
```
i=2
print didrate[i]
```

```
[1 1 1 1 1 1]
```

In [10]:
```
idx = where(didrate[i] == 1)[0]                          #check for only those
 movies whose didrate=1 (rating has been done)
print idx
```

```
[0 1 2 3 4 5]
```

In [11]:
```
ratings_mean=zeros(shape=(10,1))                        #initialize
ratings_mean[i] = mean(ratings[i, idx])                 #calculate mean for part
icular movie
print ratings_mean[i]
```

```
[ 5.16666667]
```

In [12]:
```
ratings_norm=zeros(shape=ratings.shape)                                    #m
ean normalize
ratings_norm[i,idx]=ratings[i,idx]-ratings_mean[i]
print ratings_norm[i]
```

```
[-1.16666667  2.83333333 -3.16666667 -0.16666667 -3.16666667  4.83333333]
```

In [13]:
```
def normalize_ratings(ratings, did_rate):
  #function for normalizing the data of all movies
    num_movies = ratings.shape[0]
  #ratings.shape[0]--->no of rows in ratings table

  #ratings.shape[1]--->no of cols in ratings table
    ratings_mean = zeros(shape = (num_movies, 1))
    ratings_norm = zeros(shape = ratings.shape)

    for i in range(num_movies):
        # Get all the indexes where there is a 1
        idx = where(did_rate[i] == 1)[0]
        #  Calculate mean rating of ith movie only from user's that gave a rat
ing
        ratings_mean[i] = mean(ratings[i, idx])
        ratings_norm[i, idx] = ratings[i, idx] - ratings_mean[i]

    return ratings_norm, ratings_mean
```

In [14]:
```
ratings,ratings_mean_eachmovie=normalize_ratings(ratings,didrate)
```

In [15]:
```
print ratings                          #full normalize ratings for each movie
```

```
[[ 0.          6.5        -2.5         0.         -2.5        -1.5       ]
 [ 0.         -2.6         3.4         3.4        -2.6        -1.6       ]
 [-1.16666667  2.83333333 -3.16666667 -0.16666667 -3.16666667  4.83333333]
 [ 0.          0.25       -1.75       -0.75        0.          2.25      ]
 [ 0.         -4.          1.          3.          1.         -1.        ]
 [ 3.         -1.          1.         -1.          0.         -2.        ]
 [ 3.75        0.75       -0.25        0.          0.         -4.25      ]
 [ 0.         -0.6         0.4        -0.6         1.4        -0.6       ]
 [ 0.          2.5         2.5        -5.5         0.5         0.        ]
 [-1.8        -0.8         0.          5.2         1.2        -3.8       ]]
```

In [16]:
```
print ratings_mean_eachmovie
```

```
[[ 3.5       ]
 [ 5.6       ]
 [ 5.16666667]
 [ 6.75      ]
 [ 6.        ]
 [ 4.        ]
 [ 5.25      ]
 [ 2.6       ]
 [ 6.5       ]
 [ 4.8       ]]
```

In [17]:
```
num_users=ratings.shape[1]
num_features=3     #no of features on which recommendation will be based for e
g(romance,comedy,action)
print num_features
```

```
3
```

In [18]:
```
movie_features=random.randn(num_movies,num_features)
user_prefs=random.randn(num_users,num_features)
initial_X_and_Theta=r_[movie_features.T.flatten(),user_prefs.T.flatten()]
```

In [19]:
```
print movie_features
```

```
[[-1.06892078 -0.57328271  2.08540204]
 [ 0.27600447  0.06936274 -0.82053844]
 [ 0.4101977  -2.37280022  0.17852349]
 [ 0.69089465  0.63727502 -0.53048404]
 [ 0.45983367  0.05119601  2.39688023]
 [ 0.9830813   0.96707565  0.04811016]
 [ 0.8302363  -1.33656491 -1.16390888]
 [ 0.35876826  0.35491785  1.02289667]
 [-0.15081003 -1.41015118  0.18667337]
 [-0.24761629 -0.81758535  0.44163665]]
```

In [20]:
```
print user_prefs
```

```
[[ 0.10144476 -0.39905349 -1.22331138]
 [-0.92974271 -0.93712073 -1.08822685]
 [ 1.34377355  1.27771397 -1.23292263]
 [-0.60827775  1.30340243 -0.37553744]
 [ 2.07093848 -1.26358859  2.05134876]
 [-2.69297923  1.65901364  0.55070318]]
```

In [21]:
```
initial_X_and_Theta.shape
```

Out[21]:
```
(48,)
```

In [22]:
```
def unroll_params(x_and_theta,num_users,num_movies,num_features):

    #finding the x and theta values separately

    #get the first (30) values out of 48
    first_30=x_and_theta[:num_movies * num_features]

    #convert the 3*10 matrix into 10*3 matrix
    x=first_30.reshape((num_features,num_movies)).transpose()

    #get the rest (18) values out of 48
    last_18=x_and_theta[num_movies * num_features :]

    #convert the 3*6 matrix into 6*3 matrix
    theta=last_18.reshape(num_features,num_users).transpose()

    return x,theta       #return the values (x and theta are as such the x and
 y parameter of the regression equation)
```

In [36]:
```
def unroll_params(X_and_theta, num_users, num_movies, num_features):
        # Retrieve the X and theta matrixes from X_and_theta, based on their d
imensions (num_features, num_movies, num_movies)
        # --------------------------------------------------------------------
----------------------------------------
        # Get the first 30 (10 * 3) rows in the 48 X 1 column vector
        first_30 = X_and_theta[:num_movies * num_features]
        # Reshape this column vector into a 10 X 3 matrix
        X = first_30.reshape((num_features, num_movies)).transpose()
        # Get the rest of the 18 the numbers, after the first 30
        last_18 = X_and_theta[num_movies * num_features:]
        # Reshape this column vector into a 6 X 3 matrix
        theta = last_18.reshape(num_features, num_users ).transpose()
        return X, theta
```

In [35]:
```
def calculate_cost(X_and_theta, ratings, did_rate, num_users, num_movies, num_
features, reg_param):
        X, theta = unroll_params(X_and_theta, num_users, num_movies, num_featu
res)

        # we multiply (element-wise) by did_rate because we only want to consi
der observations for which a rating was given
        cost = sum( (X.dot( theta.T ) * did_rate - ratings) ** 2 ) / 2
        # '**' means an element-wise power
        regularization = (reg_param / 2) * (sum( theta**2 ) + sum(X**2))
        return cost + regularization
```

In [25]:
```
print initial_X_and_Theta
```

```
[-1.06892078  0.27600447  0.4101977   0.69089465  0.45983367  0.9830813
  0.8302363   0.35876826 -0.15081003 -0.24761629 -0.57328271  0.06936274
 -2.37280022  0.63727502  0.05119601  0.96707565 -1.33656491  0.35491785
 -1.41015118 -0.81758535  2.08540204 -0.82053844  0.17852349 -0.53048404
  2.39688023  0.04811016 -1.16390888  1.02289667  0.18667337  0.44163665
  0.10144476 -0.92974271  1.34377355 -0.60827775  2.07093848 -2.69297923
 -0.39905349 -0.93712073  1.27771397  1.30340243 -1.26358859  1.65901364
 -1.22331138 -1.08822685 -1.23292263 -0.37553744  2.05134876  0.55070318]
```

In [54]:
```
from scipy import optimize

reg_param=50

minimizedcost_and_optimal_params=optimize.fmin_cg(calculate_cost,fprime=calcul
ate_gradient,x0=initial_X_and_Theta,
                                                  args=(ratings,didrate,num_use
rs,num_movies,num_features,reg_param),
                                                  maxiter=100,disp=True,full_ou
tput=True
                                                  )
```

```
Optimization terminated successfully.
         Current function value: 163.766667
         Iterations: 8
         Function evaluations: 20
         Gradient evaluations: 20
```

In [59]:
```
cost,optimal_movie_features_and_user_prefs=minimizedcost_and_optimal_params[1]
 minimizedcost_and_optimal_params[0]
```

In [61]:
```
movie_features,user_prefs=unroll_params(optimal_movie_features_and_user_prefs,n
um_users,num_movies,num_features)
```

In [62]:
```
print movie_features
```

```
[[  1.11491130e-07   7.31428268e-08   1.15212939e-07]
 [ -1.26282877e-07  -9.17013614e-08   9.72658895e-08]
 [  1.30346806e-07   5.09304990e-08   5.75342152e-08]
 [  8.57766914e-08   5.90351062e-08  -2.46765738e-08]
 [ -1.97306776e-07  -6.62987231e-08   7.77488710e-08]
 [ -2.84364631e-08  -3.68467140e-08   5.82964329e-08]
 [ -7.77103248e-08  -1.58993415e-07   6.91402995e-09]
 [ -2.97745191e-08   2.65169654e-08  -1.33243334e-08]
 [  1.19274091e-07   5.75733670e-08  -5.54994121e-08]
 [ -2.21046396e-07  -9.92915419e-08  -5.63526068e-08]]
```

In [63]:
```
print user_prefs
```

```
[[  1.15226788e-07  -4.49999302e-08   6.74390672e-08]
 [  8.75041498e-08   1.00306175e-07  -1.26220045e-07]
 [  5.33293561e-08  -4.78236049e-08  -1.94862041e-08]
 [ -2.08975221e-07  -9.48071110e-08   1.10018116e-09]
 [ -1.12971608e-07  -5.77934607e-09  -4.47918474e-08]
 [  1.50774348e-08   1.43045127e-07   3.23575274e-08]]
```

In [64]:
```
all_predictions=movie_features.dot(user_prefs.T)
```

In [65]:
```
print all_predictions
```

```
[[  1.73251958e-14   2.55043133e-15   2.02733686e-16  -3.01065886e-14
   -1.81786504e-14   1.58717310e-14]
 [ -3.86509457e-15  -3.25253936e-14  -4.24443784e-15   3.51909435e-14
    1.04396348e-14  -1.18741710e-14]
 [  1.66076286e-14   9.25255872e-15   3.39450770e-15  -3.20045280e-14
   -1.75968970e-14   1.11123201e-14]
 [  5.56303185e-15   1.65420804e-14   2.23199688e-15  -2.35492997e-14
   -8.92620576e-15   8.93950380e-15]
 [ -1.45082767e-14  -3.37287990e-14  -8.86662973e-15   4.76033554e-14
    1.91907114e-14  -9.94282806e-15]
 [  2.31291432e-15  -1.35424399e-14  -8.90331767e-16   9.49998331e-15
    8.14257951e-16  -3.81316336e-15]
 [ -1.33334279e-15  -2.36206864e-14   3.32466849e-15   3.13208454e-14
    9.38824617e-15  -2.36911847e-14]
 [ -5.52266440e-15   1.73621936e-15  -2.59635213e-15   3.69348065e-15
    3.80724609e-15   2.91305682e-15]
 [  7.40994428e-15   2.32170804e-14   4.68891739e-15  -3.04447536e-14
   -1.13214011e-14   8.23811316e-15]
 [ -2.48027209e-14  -2.21892031e-14  -5.94168410e-15   5.55447657e-14
    2.80699444e-14  -1.93594148e-14]]
```

In [69]:
```
predictions_for_rajeev=all_predictions[ :,0:1 ] + ratings_mean_eachmovie
print predictions_for_rajeev
```

```
[[ 3.5       ]
 [ 5.6       ]
 [ 5.16666667]
 [ 6.75      ]
 [ 6.        ]
 [ 4.        ]
 [ 5.25      ]
 [ 2.6       ]
 [ 6.5       ]
 [ 4.8       ]]
```

In [ ]:
```
```