SD211 TP Recommendation System
WEI CHEN
weichen@whu.edu.cn

1. Model presentation

Question 1.1 Obtaining the data

```
filename = 'ml-100k/u.data'
R, mask=tools.load_movielens(filename, minidata=False)
```

`minidata` is an option of the `load_movielens` function which make the function only return the first 100 lines and 200 columns of R and mask. This option allows to reduce the volume of data, which will be used to limit the time cost of following processing.

Question 1.2 Data dimension

```
print(R.shape)#get the number of users and films
print(np.sum(mask))#get the total number of rating
```

The data set consists of 100,000 ratings {1,2,3,4,5} from 943 users on 1682 movies.

Question 1.3 Convexity and lipschitz

The objective function is written as

$$g(P,Q) = \frac{1}{2}\|1_K \circ (R - QP)\|_F^2 + \frac{\rho}{2}\|Q\|_F^2 + \frac{\rho}{2}\|P\|_F^2$$

Since the Frobenius norm can be rewritten as

$$\|A\|_F^2 = tr(A^T A)$$

Thus

$$g(P,Q) = \frac{1}{2}tr\left(1_K^T 1_K \circ (R^T R + P^T Q^T QP - R^T QP - P^T Q^T R)\right) + \frac{\rho}{2}tr(Q^T Q) + \frac{\rho}{2}tr(P^T P)$$

$$\nabla_P g = \frac{1}{2}\left(2Q^T(QP \circ 1_K) - 2Q^T(R \circ 1_K)\right) + \rho P = Q^T((QP - R) \circ 1_K) + \rho P$$

$$\nabla_Q g = ((QP - R) \circ 1_K)P^T + \rho Q$$

And $g(P,Q)$ is twice differentiable, and we calculate its Hessian matrix

$$Hessian = \nabla^2 g = \begin{bmatrix} Q^T Q + \rho I & 2QP - R \\ 2QP - R & PP^T + \rho I \end{bmatrix}$$

H is symmetric, and its determinant is not necessarily positive, so H is not positive definite. Thus $g(P,Q)$ is not convex.

If the gradient is lipschitzien, it exits a constant C such that for all P, Q, P', Q' there is

$$\|\nabla g(P,Q) - \nabla g(P',Q')\| \leq L\|(P,Q) - (P',Q')\|$$

and $L = sup\|\nabla^2 g(P,Q)\|$

Since we have $\nabla_{P,P}^2 g = Q^T Q + \rho I$ and $\nabla_{Q,Q}^2 g = PP^T + \rho I$, so $\|\nabla^2 g(P,Q)\|$ has no upper bound. Thus the gradient∇g is not lipschitzien.

2. Find P with fixed Q_0

Question 2.1 Gradient of $g(P)$

$$g(P) = \frac{1}{2}\|1_K \circ (R - Q^0 P)\|_F^2 + \frac{\rho}{2}\|Q^0\|_F^2 + \frac{\rho}{2}\|P\|_F^2$$
$$\nabla g(P) = (Q^0)^T((Q^0 P - R) \circ 1_K) + \rho P$$
$$\nabla^2 g(P) = (Q^0)^T Q^0 + \rho I$$

The matrix $(Q^0)^T Q^0 + \rho I$ is positive definite, so $g(P)$ is convex.
And $\nabla^2 g(P)$ is constant when Q^0 is fixed, so $\|\nabla^2 g(P)\|$ has its upper bound, and there exist a constant L_0 such that

$$\|\nabla g(P) - \nabla g(P')\| \leq L_0 \|(P) - (P')\|$$

where $L_0 = \|\nabla^2 g(P)\| = \|(Q^0)^T Q^0\|_F + \rho$

Question 2.2 Gradient of $g(P)$

```
def objective(P, Q0, R, mask, rho):
    tmp = (R - Q0.dot(P)) * mask
    val = np.sum(tmp ** 2)/2. + rho/2. * (np.sum(Q0 ** 2) + np.sum(P ** 2))
    grad_P = np.transpose(Q0).dot((Q0.dot(P)-R)*mask) + rho*P
    return val, grad_P
```

We want to verify the function defined above is correct, with the help of the function `scipy.optimize.check_grad`.

```
def g_PQ(P,Q):
    P=P.reshape(vt.shape)
    return tools.total_objective(P, Q, R, mask, rho)[0]


def grad_P(P,Q):
    P=P.reshape(vt.shape)
    return np.ravel(tools.total_objective(P, Q, R, mask, rho)[1])


def grad_Q(Q,P):
    Q=Q.reshape(u.shape)
    return np.ravel(tools.total_objective(P, Q, R, mask, rho)[2])
```

```
u,s,vt = svds(R,7)
Q0=u
P0=vt
P0=np.ravel(P0)
rho=0.2
check_grad(g_P, grad_P, P0, Q, epsilon=1e-6)
```

This function will return the 2-norm of the difference between `grad_P(P0, Q)` and the finite difference approximation of *grad* using `g_p`. The result we obtain is

`0.021928157886692275`

This difference is small enough, which indicates the gradient of P calculated by function `grad_P` is verified.

Question 2.3 Gradient descent with invariant pace

Use the gradient of *g(P)*defined in the previous question, update P_k with iteration as following pattern:

$$P_{k+1} = P_k - \gamma_k \nabla g(P_k)$$

where γ_k is the pace length, and in the case of invariant pace it is constant.

```
def gradient(g,P0,gamma,epsilon):
    P=P0
    grad_P=tools.objective(P, Q0, R, mask, rho)[1]
    G=list([])
    C=list([])
    while np.sqrt(np.sum(grad_P**2))>epsilon:
        P=P-gamma*grad_P
        grad_P=tools.objective(P, Q0, R, mask, rho)[1]
        G.append(np.sqrt(np.sum(grad_P**2)))
        C.append(tools.objective(P, Q0, R, mask, rho)[0])
    return P,G,C
```

Question 2.4 Minimise $g(P)$ with Gradient descent

we set the pace length as

$$\gamma_k = \frac{1}{L_0}$$

where $L_0 = \|(Q^0)^T Q^0\|_F + \rho$ is the lipschitz constant of $\nabla g(P)$. We've proved its convergence in Theorem 3.3.1 in the course.

```
u,s,vt = svds(R,7)
Q0=u
P0=vt
rho=0.2
L0=np.sqrt(np.sum(np.transpose(Q0).dot(Q0)))+rho
gamma=1/L0
P_opt,G,C=gradient(0,P0,gamma,1)
plt.figure()
plt.plot(G,linestyle='--',marker='o')
plt.xlabel('iteration')
plt.ylabel('F-norm of grad_P')
plt.grid()
plt.figure()
plt.plot(C,linestyle='--',marker='s')
plt.xlabel('iteration')
plt.ylabel('Objective value')
```

```
plt.grid()
plt.show()
```

In order to verify the convergence of this algorithm, we show the evolution of the norm of the gradient $\|\nabla g(P_k)\|_F$ and the evolution of the objective value $g(P_k)$ for each iteration. From the figure below, we see both $\|\nabla g(P_k)\|_F$ and $g(P_k)$ are decreasing with the evolution, which verifies the convergence of the algorithm.

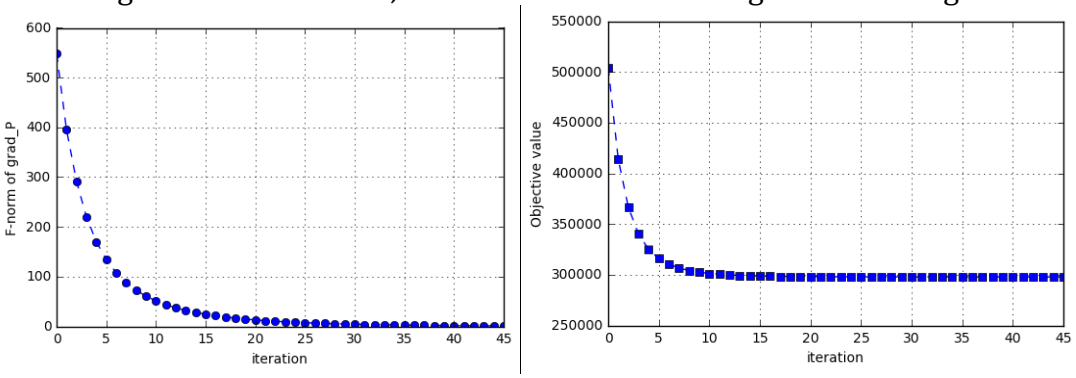

Fig. Convolution of $\|\nabla g(P_k)\|_F$ and $g(P_k)$in each iteration

3. Algorithm Refinement with fixed Q_0

Question 3.1 Gradient descent with line search

The gradient method with constant pace need to compute the Lipschitz constant of the gradient, which may require much work. Even more, for problem (1) the gradient of $g(P,Q)$ is not lipschitzien, so we should consider a new method free from computing Lipschitz constant.

The idea of line search is to choose the pace γ_k adaptively using local information. In exact line search we take

$$\gamma_k = argmin_\gamma \, g(x_k - \gamma \nabla g(x_k))$$

We apply the function `scipy.optimize.line_search` to solve this optimisation problem.

```
def linesearch_gradient(P0,epsilon):
    P=P0
    grad=tools.objective(P, Q, R, mask, rho)[1]
    G=list([])
    C=list([])
    while np.sqrt(np.sum(grad**2))>epsilon:
        gamma=line_search(g_PQ, grad_P, np.ravel(P), -grad_P(np.ravel(P),Q),
         args=(Q,))[0]
        P=P-gamma*grad
        grad=tools.objective(P, Q, R, mask, rho)[1]
        G.append(np.sqrt(np.sum(grad**2)))
        C.append(tools.objective(P, Q, R, mask, rho)[0])
    return P,G,C
```

```
u,s,vt = svds(R,7)
Q0=u
```

```
P0=vt
rho=0.2


P_opt,G,C=linesearch_gradient(P0,1)
plt.figure()
plt.plot(G,linestyle='--',marker='o')
plt.xlabel('iteration')
plt.ylabel('F-norm of grad_P')
plt.grid()
plt.figure()
plt.plot(C,linestyle='--',marker='s')
plt.xlabel('iteration')
plt.ylabel('Objective value')
plt.grid()
plt.show()
```

Similarly, we show the convolution of $\|\nabla g(P_k)\|_F$ for each iteration.

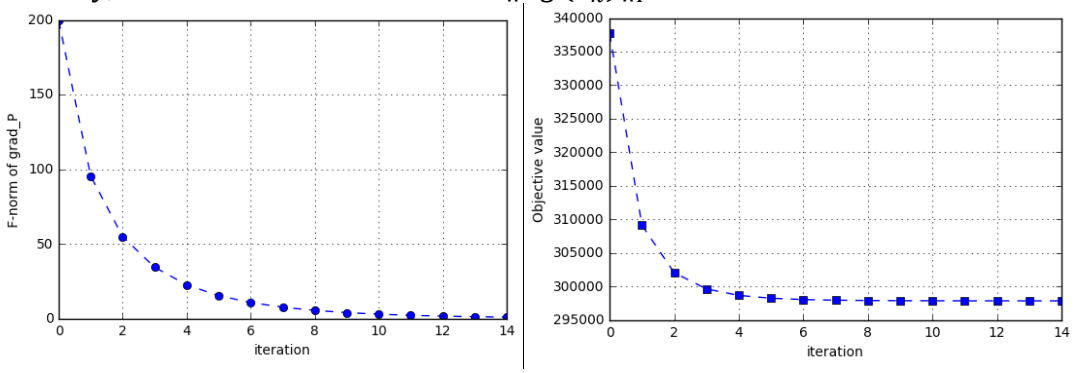

Fig. Convolution of $\|\nabla g(P_k)\|_F$ and $g(P_k)$ with line search

Question 3.2 Conjugate gradient method

The nonlinear conjugate gradient method is generally used to find the local minimum of a nonlinear function using its gradient alone. It works when the function is approximately quadratic near the minimum, which is the case when the function is twice differentiable at the minimum and the second derivative is non-singular there.

Since $g(P)$ is continuously twice differentiable and is non-singular at the minimum, so the conjugate gradient method can be applied to the problem when Q is fixed.

The principle of the algorithm is described as follows:

Initialise

$k = 0$,

P^0,

$d_0 = -\nabla g(P^0)$,

ϵ

repeat

- choose $\gamma_k = argmin\ g(P_k - \gamma \nabla g(P_k))$

- $P_{k+1} = P_k + \gamma_k d_k$
- $b_k = \frac{\|\nabla g(P_{k+1})\|^2}{\|\nabla g(P_k)\|^2}$
- $d_{k+1} = -\nabla g(P_{k+1}) + b_k \cdot d_k$
- k=k+1

until $\|\nabla g(P_k)\|_F < \epsilon$

```
def conj_gradient(P,Q,epsilon):
    grad=tools.total_objective(P, Q, R, mask, rho)[1]
    d=-grad
    G=list([])
    C=list([])
    while np.sqrt(np.sum(grad**2))>epsilon:
        gamma=line_search(g_PQ, grad_P, np.ravel(P), -grad_P(np.ravel(P),Q),
         args=(Q,))[0]
        P=P+gamma*d
        grad_old=grad
        grad=tools.total_objective(P, Q, R, mask, rho)[1]
        b=np.sum(grad**2)/np.sum(grad_old**2)
        d=-grad+b*d
        G.append(np.sqrt(np.sum(grad**2)))
        C.append(tools.objective(P, Q, R, mask, rho)[0])
    return P,G,C
```

```
u,s,vt = svds(R,7)
Q=u
P0=vt
P_opt,G,C=conj_gradient(P0,Q,1)
plt.figure()
plt.plot(G,linestyle='--',marker='o')
plt.xlabel('iteration')
plt.ylabel('F-norm of grad_P')
plt.grid()
plt.figure()
plt.plot(C,linestyle='--',marker='s')
plt.xlabel('iteration')
plt.ylabel('Objective value')
plt.grid()
plt.show()
```

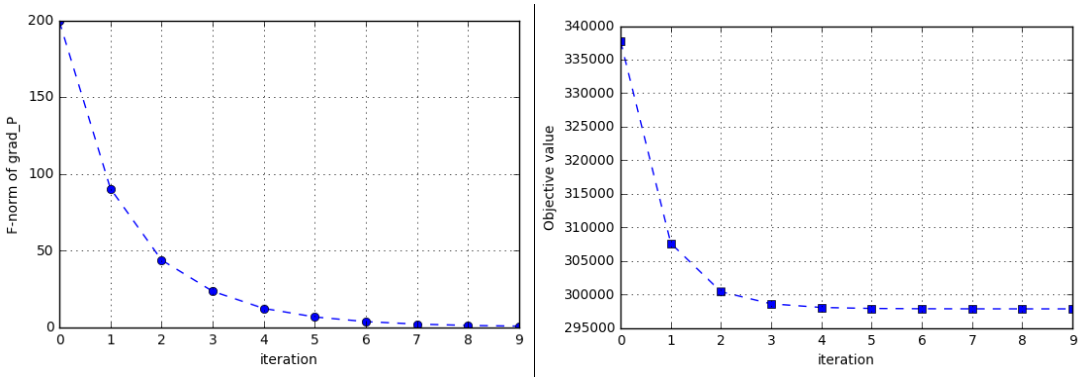

Fig. Convolution of $\|\nabla g(P_k)\|_F$ and $g(P_k)$ with conjugate gradient method

Question 3.3 Comparison

With the same stop condition, we see the number of iteration required for convergence for these three methods:

	gradient with constant pace length	gradient with line search	conjugate gradient
number of iteration required for convergence	45	14	9
Objective value after convergence	297828	297828	297827

We can conclude that the three method above converge to the same solution, but with different convergence speed:
gradient with constant pace length< gradient with line search< conjugate gradient

4. Resolution of the complete problem

4.1 Solve problem (1) with the gradient method with line search

```
def g_PQvec(PQvec):
    return tools.total_objective_vectorized(PQvec, R, mask, rho)[0]


def grad_PQ(PQvec):
    return tools.total_objective_vectorized(PQvec, R, mask, rho)[1]


def linesearch_gradient_PQ(P0, Q0, epsilon):
    PQvec=np.concatenate([P0.ravel(), Q0.ravel()])
    grad=grad_PQ(PQvec)
    G=list([])
    C=list([])
    while np.sqrt(np.sum(grad**2))>epsilon:
        gamma=line_search(g_PQvec, grad_PQ, PQvec, -grad_PQ(PQvec) )[0]
        PQvec=PQvec-gamma*grad
```

```
        grad=grad_PQ(PQvec)
        G.append(np.sqrt(np.sum(grad**2)))
        C.append
        (tools.total_objective_vectorized(PQvec, R, mask, rho)[0])
    return PQvec,G,C
```

```
u,s,vt = svds(R,7)
Q0=u
P0=vt
rho=0.2
PQ_opt,G=linesearch_gradient_PQ(P0,Q0,100)
plt.plot(G,linestyle='--',marker='o')
plt.xlabel('iteration')
plt.ylabel('F-norm of grad_P')
plt.grid()
plt.show()
```

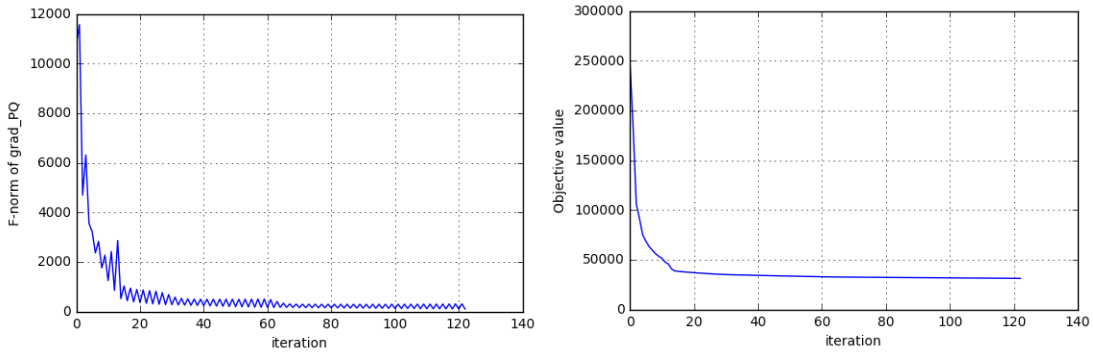

Fig. Convolution of $\|\nabla g(P_k, Q_k)\|_F$ and $g(P_k, Q_k)$ with line search

4.2 Convergence of alternating least square method

Since $P_k = argmin_P g(P, Q_{k-1})$, we have $g(P_k, Q_{k-1}) \leq g(P_{k-1}, Q_{k-1})$
Similarly from $Q_k = argmin_Q g(P_k, Q)$, we have $g(P_{k-1}, Q_{k-1}) \leq g(P_{k-1}, Q_{k-2})$
From the two inequality above we have

$$g(P_k, Q_{k-1}) \leq g(P_{k-1}, Q_{k-1}) \leq g(P_{k-1}, Q_{k-2})$$

In the same way, we can prove that

$$g(P_k, Q_k) \leq g(P_{k-1}, Q_{k-1})$$

This indicates that the objective functions decrease with the iteration, and finally converge.

4.3 Alternating Least square

Optimise P and Q alternatively, by minimising the following two equations

$$P_k = argmin \frac{1}{2}\|1_K \circ (R - Q_{k-1}P)\|_F^2 + \frac{\rho}{2}\|Q_{k-1}\|_F^2 + \frac{\rho}{2}\|P\|_F^2$$

$$Q_k = argmin \frac{1}{2}\|1_K \circ (R - QP_k)\|_F^2 + \frac{\rho}{2}\|Q\|_F^2 + \frac{\rho}{2}\|P_k\|_F^2$$

The principle of the algorithm is shown as follows.

Initialise

$k = 0$,
P_0
Q_0
ϵ

for $k \geq 1$ do
  for i=1,…,1682
    $\Lambda_i = diag(1_{K_{\cdot,i}})$
    $P_{\cdot,i}^{(k)} = (Q_{k-1}^T \Lambda_i Q_{k-1} + \rho I)^{-1} Q_{k-1}^T (R_{\cdot,i} \Lambda_i)$
  for j=1,…,943
    $\Lambda_u = diag(1_{K_{u,\cdot}})$
    $Q_{u,\cdot}^{(k)} = (R_{u,\cdot} \Lambda_u) P_k^T (P_k \Lambda_u P_k^T + \rho I)^{-1}$
- k=k+1

until $|g(P_k, Q_k) - g(P_{k-1}, Q_{k-1})| < \epsilon$

```
u,s,vt = svds(R,7)

Q=u

P=vt

C=list([])

epsilon=100

rho=0.2

k=0

while k<2 or np.abs(C[-1]-C[-2])>10:

    for i in range(P.shape[1]):

        P[:,i]=np.linalg.inv (Q.T.dot( np.diag(mask[:,i]) ).dot(Q)+rho*np.ey
        e(Q.shape[1])).dot(Q.T).dot(R[:,i].dot(np.diag(mask[:,i])))

    for u in range(Q.shape[0]):

        Q[u,:]=R[u,:].dot(np.diag(mask[u,:])).dot(P.T).dot(np.linalg.inv(P.d
        ot(  np.diag(mask[u,:])  ).dot(P.T)+rho*np.eye(P.shape[0])))

    C.append(tools.total_objective(P, Q, R, mask, rho)[0])

    k=k+1

plt.plot(C)

plt.xlabel('iteration')

plt.ylabel('Objective value')

plt.grid()

plt.show()
```

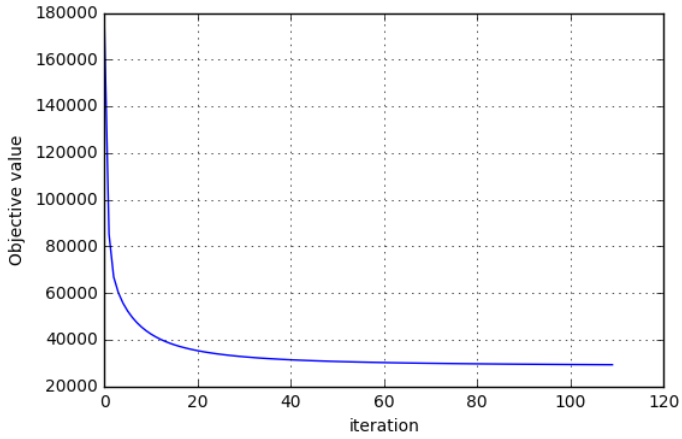

Fig. Convolution of $g(P_k, Q_k)$ with alternating least square

4.4 Comparison

	gradient with line search	alternating least square
NMSE of P,Q between two methods	2.066	
NMSE of R between two methods	1.366	
number of iteration required for convergence	123	110
Objective value after convergence	31148	29198
Calculation time	195s	5939s

We define the normalized mean square error (NMSE) to measure the difference of P, Q and R,

$$NMSE(X, X') = \frac{1}{N}\|X - X'\|^2$$

From the NMSE we can see there is great difference between the results of these two methods.

As for the performance, alternating least square convergence a little faster than gradient method, and

4.4 Recommendation

We estimate R by

$$\hat{R} = \hat{Q}\hat{P}$$

If we recommend a film to user 449, we will choose the film which he/she hasn't rated but would give an estimated highest rating.

$$i_{recommend} = argmax_i \, \hat{R}_{449,i} * (1 - 1_K)$$

```
P_opt=PQ_opt[:P0.shape[0]*P0.shape[1]].reshape(P0.shape)
Q_opt=PQ_opt[P0.shape[0]*P0.shape[1]:].reshape(Q0.shape)
R_est=Q_opt.dot(P_opt)
print(np.argmax(R_est[449,:]*(1-mask)[449,:]))
print(np.max(R_est[449,:]))
```

```
plt.figure(figsize=(15,6))
plt.bar(range(R.shape[1]),R_est[449,:]*(1-mask)[449,:])
plt.xlabel('Film')
plt.ylabel('Rate of user 449')
plt.grid()
plt.show()
```

We use both gradient method with line search and alternating least square method to estimate P and Q, and get the recommendation for user 449. (We notice that in the data sets, users and films are numbered consecutively from 1.)

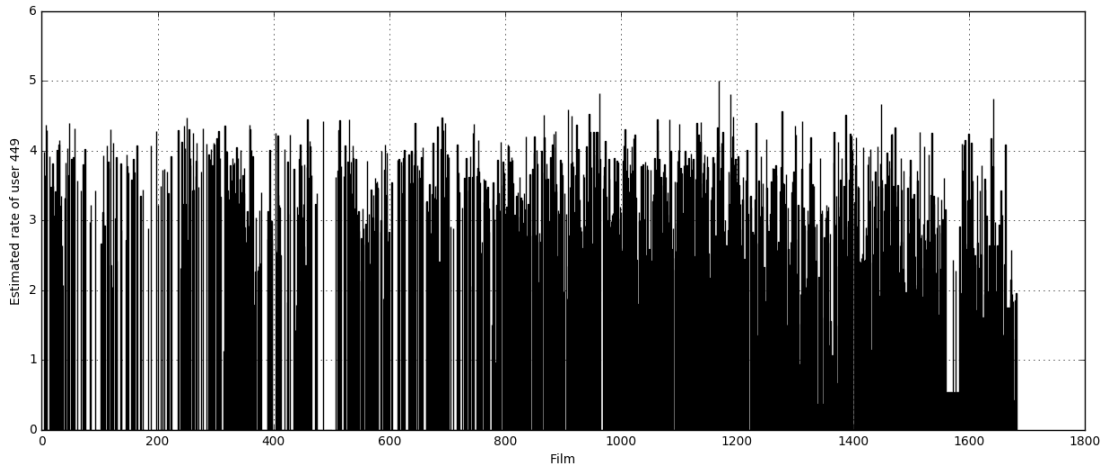

Fig. Estimate by gradient with line search

By gradient method with line search, we get the estimate of R, and we recommend **film 1449** to user 449.
We notice that in the data sets, users and films are numbered consecutively from 1. In the data set `u.item` we find the information of **film 1449**.

Fig. Film recommended by gradient with line search

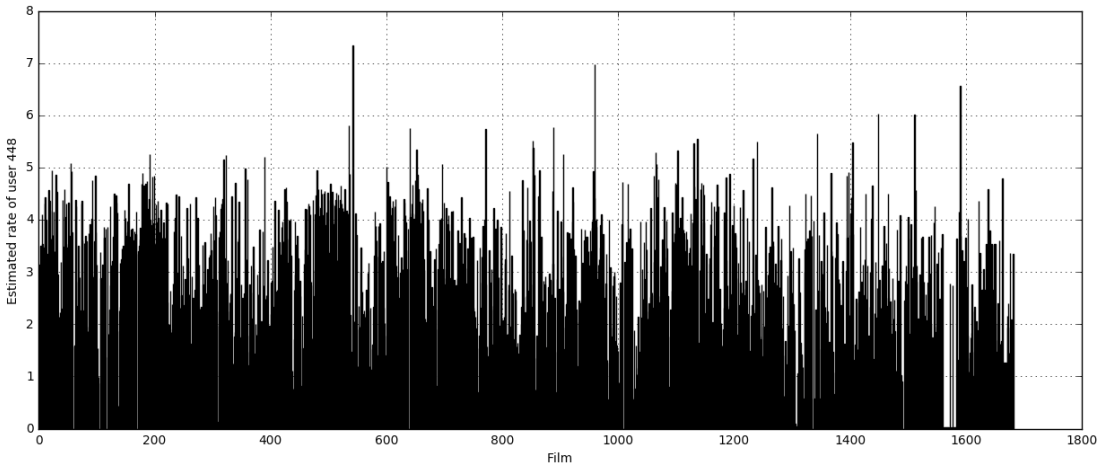

Fig. Estimate by alternating least square

By alternating least square method, we recommend **film 124** to user 449. We can conclude that these two methods will lead to different recommendation result.

Fig. Film recommended by alternating least square