Q02

July 19, 2020

[1]:
```
from scipy.io import loadmat
import numpy as np
```

[3]:
```
ratings = loadmat('ratings.mat')["M0"]
missing = loadmat('ratings_missing.mat')["M1"]
```

[4]:
```
mask = missing != 0
m,n = ratings.shape
x = cp.Variable((m,n))
objective = cp.Minimize(cp.normNuc(x))
constraints = [x[mask] == ratings[mask], x>=1]
prob = cp.Problem(objective, constraints)
result = prob.solve(solver='SCS', verbose=True)
```

```
----------------------------------------------------------------------------
        SCS v2.0.2 - Splitting Conic Solver
        (c) Brendan O'Donoghue, Stanford University, 2012-2017
----------------------------------------------------------------------------
Lin-sys: sparse-indirect, nnz in A = 75150, CG tol ~ 1/iter^(2.00)
eps = 1.00e-04, alpha = 1.50, max_iters = 5000, normalize = 1, scale = 1.00
acceleration_lookback = 20, rho_x = 1.00e-03
Variables n = 45150, constraints m = 75150
Cones:  primal zero / dual free vars: 10000
        linear vars: 20000
        sd vars: 45150, sd blks: 1
Setup time: 5.53e-02s
----------------------------------------------------------------------------
 Iter | pri res | dua res | rel gap | pri obj | dua obj | kap/tau | time (s)
----------------------------------------------------------------------------
     0| 2.89e+21  3.75e+21  1.00e+00 -1.12e+25  1.71e+25  1.58e+24  8.08e-02
   100| 1.44e-03  2.07e-03  1.98e-03  9.64e+02  9.61e+02  4.51e-13  3.88e+00
   200| 1.19e-04  1.64e-04  6.01e-05  9.66e+02  9.66e+02  3.16e-13  7.17e+00
   240| 4.60e-05  6.13e-05  2.47e-06  9.65e+02  9.65e+02  9.10e-13  8.45e+00
----------------------------------------------------------------------------
Status: Solved
Timing: Solve time: 8.45e+00s
        Lin-sys: avg # CG iterations: 1.00, avg solve time: 8.74e-04s
        Cones: avg projection time: 2.25e-02s
```

```
        Acceleration: avg step time: 9.50e-03s
----------------------------------------------------------------------------
Error metrics:
dist(s, K) = 3.9348e-07, dist(y, K*) = 6.1075e-08, s'y/|s||y| = -6.0854e-10
primal res: |Ax + s - b|_2 / (1 + |b|_2) = 4.6023e-05
dual res:   |A'y + c|_2 / (1 + |c|_2) = 6.1260e-05
rel gap:    |c'x + b'y| / (1 + |c'x| + |b'y|) = 2.4652e-06
----------------------------------------------------------------------------
c'x = 965.4403, -b'y = 965.4450
============================================================================
```

[7]:
```python
np.sum(np.abs(ratings-x.value)/ratings)
```

[7]: 344.4646372724289

[5]:
```python
A = missing
n1,n2 = A.shape
r = 2
u, s, vh = np.linalg.svd(A)
s[r:] = 0

mask = missing == 0
m = (~mask).sum()
Y = np.zeros((n1, n2))
delta = n1*n2/m
tau = 500
# %Iterations
vec = []
err = []
for i in range(500):
    u, s, vh = np.linalg.svd(Y)
    s_t = np.maximum(s-tau, 0)
    Z = (u[:, :n2]*s_t)@vh
    P = missing-Z
    P[mask] = 0
    Y0 = Y.copy()
    Y = Y0 + delta*P
```

[6]:
```python
np.sum(np.abs(ratings-Z)/ratings)
```

[6]: 391.8841094041523

[7]:
```python
def mat_comp(A,tau,delta):
    n1,n2 = A.shape
    r = 2
    u, s, vh = np.linalg.svd(A)
```

```
    s[r:] = 0
    mask = missing == 0
    m = (~mask).sum()
    Y = np.zeros((n1, n2))
    for i in range(500):
        u, s, vh = np.linalg.svd(Y)
        s_t = np.maximum(s-tau, 0)
        Z = (u[:, :n2]*s_t)@vh
        P = A-Z
        P[mask] = 0
        Y0 = Y.copy()
        Y = Y0 + delta*P
    return Z
```

[8]:
```
iters = [(.1,50),(2,50),(.1,500),(2,500)]

for iter in iters:
    delta,tau = iter
    Z = mat_comp(missing,tau,delta)
    error = np.sum(np.abs(ratings-Z)/ratings)
    print(f"tau: {tau} delta: {delta} Error: {error}")
```

```
tau: 50 delta: 0.1 Error: 4533.8829608937085
tau: 50 delta: 2 Error: 4555.1019317726505
tau: 500 delta: 0.1 Error: 931.1822789332377
tau: 500 delta: 2 Error: 391.8841094041523
```

0.0.1 Part C

The singular value thresholding algorithm executed much faster than the cvxpy implementation of nuclear norm minimization with the same reconstruction error, once tau and delta were known. Terefore it is the better method.