Exercise 09: Matrix Factorization

A sparse matrix factorization:

In the case of a single factor, $K = 1$, so that $\hat{X} = duv^T$. This gives us a rank-1 approximation to the original matrix. The Witten et. al. paper proposes to estimate u and v by solving the following optimization problem:

$$\underset{u \in \mathbb{R}^N, v \in \mathbb{R}^P}{minimize} \ \|X - duv^T\|_F^2$$
$$\text{subject to } \|u\|_2^2 = 1 \ , \ \|v\|_2^2 = 1, \ \|u\|_1 \leq \lambda_u \ , \ \|v\|_1 \leq \lambda_v$$

This is equivalent to:

$$\underset{u \in \mathbb{R}^N, v \in \mathbb{R}^P}{maximize} \ u^T xv$$
$$\text{subject to } \|u\|_2^2 = 1 \ , \ \|v\|_2^2 = 1, \ \|u\|_1 \leq \lambda_u \ , \ \|v\|_1 \leq \lambda_v$$

According the the paper, The following iterative algorithm is used to optimize the criterion for the rank-1 PMD:

Step 1. Initialize v to have $L2$-norm 1.

Step 2. Iterate until convergence:

(a) $u = argmax_u u^T Xv$ subject to $\|u\|_1 \leq \lambda_u$ and $\|u\|_2^2 \leq 1$.

The solution $u = \frac{S(Xv, \Delta_u)}{\|S(Xv, \Delta_u)\|_2}$, with $\Delta_u = 0$ if this results in $|u| \leq \lambda_u$; otherwise, Δ_u is chosen s.t. $|u| = \lambda_u$. Here S denotes the soft thresholding operator.

$S(Xv, \Delta_u) = sgn(Xv)(|Xv| - \Delta_u)_+$.

(b) $v = argmax_v u^T Xv$ subject to $\|v\|_1 \leq \lambda_v$ and $\|v\|_2^2 \leq 1$.

The solution $v = \frac{S(X^T u, \Delta_v)}{\|S(X^T u, \Delta_v)\|_2}$, with $\Delta_v = 0$ if this results in $|v| \leq \lambda_v$; otherwise, Δ_v is chosen s.t. $|v| = \lambda_v$. Here S denotes the soft thresholding operator.

$S(X^T u, \Delta_v) = sgn(X^T u)(|X^T u| - \Delta_u)_+$.

Step 3. $d = u^T Xv$.

Computation of K factors of PMD:

Step 1. Let $X^1 = X$

Step 2. For $k = 1, 2, \cdots, K$:

(a) Find u_k, v_k, and d_k by appling the single-factor PMD algorithm to data X^k.

(b) $X^{k+1} = X^k - d_k u_k v_k^T$.

Code for rank-1 & rank-k PMD:

```
#### Generate a simple test version X ####
library(MASS)
N=50
p=5 # Number of indept. variables, i.e. the length of vector beta.
mu.test=rep(1,p)
sigma.test=diag(0.5, p)
x.test=mvrnorm(N, mu=mu.test, Sigma=sigma.test) # Indept. R.V.

#### Functions ####
prox=function(ut, gamma=1, lambda){

prox=sign(ut) * pmax( ( abs(ut) -  gamma* lambda ), 0)

return(prox)
}

Delta= function( a, c  ){
Delta.lower = 0
Delta.upper = max(abs(a))

if(norm(a, type = "2")==0 || sum(abs(a/sqrt(sum(a^2)))) <= c) {
return(0)
}

i=1
while(i<200){
su = prox(a, gamma=1, (Delta.lower + Delta.upper)/2)
if(sum(abs(su/sqrt(sum(su^2)))) < c){
Delta.upper = (Delta.lower + Delta.upper)/2
} else {
```

```
Delta.lower = (Delta.lower + Delta.upper)/2
}
if((Delta.upper - Delta.lower) < 1e-6) {
return((Delta.lower + Delta.upper)/2)
}
i=i+1
}
warning("Didn't quite converge")
Delta = (Delta.lower + Delta.upper)/2

return( Delta )
}

PMD.1=function( X, c.u, c.v, maxiter ){

# Initialize v to have L2 norm=1 #
v = rep(1, p)
v = v / norm(v, type = "2")
u = rep(0, p)

v.t=v
t=1
while( t < maxiter ){
Delta.u.t=Delta( X %*% v.t, c.u )
u.t.prox = prox( ut= X %*% v.t, gamma=1, lambda= Delta.u.t )
u.t = u.t.prox / norm(u.t.prox, type = "2" )
u= cbind(u, u.t)

Delta.v.t=Delta( t(X) %*% u.t, c.v )
v.t.prox = prox( ut= t(X) %*% u.t, gamma=1, lambda= Delta.v.t )
v.t = v.t.prox / norm(v.t.prox, type = "2" )
v= cbind(v, v.t)
```

```
t=t+1
}

u.final= u[,t]
v.final= v[,t]
d= t(u.final) %*% X %*% v.final

return( list( iter=t, u=u, v=v, u.final=u.final, v.final=v.final ,d=d ) )
}

PMD.k = function(X, k, c.u, c.v, maxiter, tol ){
if(k == 1){
return(PMD.1(X, c.u, c.v, maxiter, tol))
}

u.PDMk=rep(0, N)
v.PDMk = rep(0,p)
d.PDMk = 0
X1 = X
for(i in 1:k){
r1 = PMD.1(X1, c.u, c.v, maxiter, tol)
d.PDMk = c(d.PDMk, r1$d)
X1 = X1 - as.numeric(r1$d) * r1$u.final %o% r1$v.final
u.PDMk = cbind(u.PDMk, r1$u.final)
v.PDMk = cbind(v.PDMk, r1$v.final)
}

return( list( X.final=X1, u.PDMk=u.PDMk, v.PDMk=v.PDMk , d.PDMk=d.PDMk ) )
}
```

```
fit.PDMk=PMD.k( X=x.test, k=2, c.u=c.u, c.v=c.v, maxiter=maxiter, tol=tol )
```

```
maxiter=50

tol=1e-4

c.u=1

c.v=1
```



```
fit.PDMk=PMD.k( X=x.test, k=2, c.u=c.u, c.v=c.v, maxiter=maxiter, tol=tol )
```