Simulation

September 19, 2019

Generating the Incomplete Matrix

Suppose $L^* \in R^{n_1 \times r}, R^* \in R^{n_2 \times r}, r \leq \min(n_1, n_2)$.

Generate L^* and R^* from the standard Normal distribution, then M^* is calculated as $M^* = L^* R^{*T} \in R^{n_1 \times n_2}$.

The missing indices are randomly selected from $[n_1] \times [n_2]$ with probability 0.1, the corresponding entries are then set to zero. Denoting the observed entries with Ω, our observation is M_{obs} with

$$[M_{obs}]_{ij} = \begin{cases} M^*_{ij}, & (i,j) \in \Omega, \\ 0, & \text{otherwise.} \end{cases}$$

Setting $n_1 = 10, n_2 = 10, r = 5$ and following the above procedure, M_{obs} is generated and displayed as following.

```r
# L: n1*r, R: n2*r, M: n1*n2
n1 = 10; r = 5; n2 = 10
missfrac = 0.1

set.seed(1983)
L_true = matrix(rnorm(n1*r), n1, r)
set.seed(831)
R_true = matrix(rnorm(n2*r), n2, r)
M_true = L_true %*% t(R_true)


Projector = function(M_complete, miss_id = imiss){
  M_miss = M_complete
  M_miss[miss_id] = 0
  return(M_miss)
}


set.seed(19)
imiss = sample(seq(n1*n2), n1*n2*missfrac, replace = F)
M_obs = Projector(M_true, imiss)

M_obs
```

```
##            [,1]       [,2]        [,3]        [,4]        [,5]
##  [1,] -0.8509757 -2.0229584  1.25382178  0.02235849 -2.48244313
##  [2,] -2.1280596  0.0000000  0.00000000 -1.52361457 -0.60182932
##  [3,]  2.2187205 -3.6296800  2.43393887  3.13315894 -0.19069610
##  [4,] -0.2707801  0.8777913 -0.80227905 -0.65758722  0.09455811
##  [5,] -0.5683552  5.4317821 -9.19010645 -1.60044613 -4.03906193
##  [6,]  0.2992269 -6.2052645  5.63717522  0.00000000 -1.48088434
##  [7,]  0.0000000  3.3930990 -3.10648786  0.02132820  0.61169981
##  [8,] -1.0358277 -2.1156629  0.00000000 -0.85499089  0.00000000
```

```
##  [9,] -1.3206974  0.4439693  0.02401604 -0.50654735 -0.82616192
## [10,] -0.1500047  3.7133849 -5.68566426 -0.30589079 -1.57317478
##              [,6]       [,7]       [,8]       [,9]      [,10]
##  [1,]  0.02623258 -1.2222788  0.2141699  0.8479329  2.2573312
##  [2,] -2.10623035  1.7812395  0.2317310 -0.5043064  1.2699727
##  [3,] -2.29920810  2.0601949 -0.6018678  0.3998622 -3.5702530
##  [4,]  0.00000000  0.0000000 -0.3131487  0.1163618  0.6731659
##  [5,]  6.01899259 -6.4467973  1.1472557  4.5196409  2.3191385
##  [6,] -1.38413114 -0.1645132 -2.4245985 -1.1526871  4.4111552
##  [7,]  4.28760873  0.0000000  0.0000000 -2.8889722  3.3469463
##  [8,] -1.69507093  1.6160353 -1.4384228 -0.3845476  1.6498730
##  [9,]  0.79553433 -1.8341109  2.5303806 -0.6774710  1.6459835
## [10,]  4.53246688 -5.3677120  2.5303575  1.0822569  1.9762316
```

Vanilla GD

Our objective is to solve for

$$\arg \min_{L\in\mathbb{R}^{n_1\times r}, R\in\mathbb{R}^{n_2\times r}} \frac{1}{2}\|\mathcal{P}_\Omega(LR^T - M_{obs})\|_F^2 \triangleq f(L,R).$$

Vanilla gradient descent algorithm gives the following update rule

$$\begin{aligned}
L_{t+1} &= L_t - \eta\nabla_L f(L_t,R_t) = L_t - \eta\ \mathcal{P}_\Omega(L_tR_t^T - M)R_t,\\
R_{t+1} &= R_t - \eta\nabla_R f(L_t,R_t) = R_t - \eta\ \mathcal{P}_\Omega(L_tR_t^T - M)^T L_t.
\end{aligned}$$

Suppose matrix $\frac{1}{p}M_{obs}$ has svd $\frac{1}{p}M_{obs} = UDV^T$, where p is the observing probability. The suggested spectral initialization (Chi et al., 2018) is

$$\begin{aligned}
L_0 &= UD^{\frac{1}{2}},\\
R_0 &= VD^{\frac{1}{2}}.
\end{aligned}$$

```r
# Initialization ----------------------------------------------------------

library(rsvd)
LR_decomp = function(M)
{
  D = diag(rsvd(M, r)$d); U = rsvd(M, r)$u; V = rsvd(M, r)$v #### Note: assume r is given.
  L0 = U %*% sqrt(D)
  R0 = V %*% sqrt(D)
  return(list(L0=L0, R0=R0))
}
```

```r
# Vanilla GD --------------------------------------------------------------

VanillaGD = function(M, L0, R0, lr, MaxIter)
{
  Lt = L0; Rt = R0;
  for (t in 1:MaxIter) {
```

```r
    L = Lt - lr * Projector(Lt%*%t(Rt)-M) %*% Rt
    R = Rt - lr * t(Projector(Lt%*%t(Rt)-M)) %*% Lt
    Lt = L
    Rt = R
  }
  return(list(Lt = Lt, Rt = Rt))
}

L0 = LR_decomp(M_obs/(1-missfrac))$L0; R0 = LR_decomp(M_obs/(1-missfrac))$R0
LR_est = VanillaGD(M_obs, L0, R0, 0.01, 10000)
L_est = LR_est$Lt; R_est = LR_est$Rt

norm(Projector(L_est%*%t(R_est)-M_obs), 'f')/norm(M_obs, 'f')
```

```
## [1] 1.079114e-07
```

```r
norm((L_est%*%t(R_est)-M_true), 'f')/norm(M_true, 'f')
```

```
## [1] 8.947059e-07
```

```r
L_true
```

```
##              [,1]         [,2]       [,3]        [,4]         [,5]
##  [1,] -0.01705205  0.133463040 -1.5804783  1.25536259 -0.137244614
##  [2,] -0.78367184 -0.927706348 -1.2597907  0.92854392 -1.664400080
##  [3,]  1.32662703  2.207440778 -1.0548884 -0.61815513  0.410246671
##  [4,] -0.23171715 -0.504477414  0.3127123 -0.06656542  0.138299087
##  [5,] -1.66372191 -0.727590759 -0.1062695  0.96360792  2.378799750
##  [6,]  1.99692302  0.593223401 -1.4651610  0.83648041 -0.002550553
##  [7,]  0.04241627  0.154716749  1.9802851  1.70499096 -0.535470806
##  [8,] -0.01241974 -0.720989534 -0.8616079  0.30285842 -0.767703360
##  [9,] -0.47278737 -0.130735800 -0.2762007  1.30924806 -0.838132433
## [10,] -0.53680130 -0.004721653  0.7957489  0.92294185  1.158754642
```

```r
L0
```

```
##              [,1]        [,2]       [,3]        [,4]       [,5]
##  [1,] -0.07419765  1.36016572 -0.6418993 -0.16949459  0.2852832
##  [2,] -0.25324914  0.34150387  0.6930583 -1.48269053  0.1443874
##  [3,] -1.33984146 -0.77496091 -1.3600175  0.73490064 -0.3833664
##  [4,]  0.23254946  0.04492612  0.2897294 -0.32766713 -0.2381277
##  [5,]  3.44548800  0.51381703 -1.0783187 -0.44235313 -0.7733421
##  [6,] -1.49262295  2.56569732 -0.2768372  0.61203201 -0.3297138
##  [7,]  1.18959029  0.28500945  1.9137603  1.14525557 -0.5881225
##  [8,] -0.51497558  0.63803334  0.3867248 -0.85200471 -0.8235840
##  [9,]  0.45898880  0.67817653  0.2101536 -0.02197248  1.4650862
## [10,]  2.34494448  0.37363541 -0.2701427  0.56264743  0.5863265
```

```r
L_est
```

```
##             [,1]       [,2]        [,3]        [,4]       [,5]
##  [1,]  0.6397439  1.2648032  0.20068118 -0.52729048  0.3788204
##  [2,] -0.6938901  0.7155203 -0.10228761 -0.93237893  1.1860699
##  [3,]  1.8521003 -0.8351893 -0.67188946  0.01845588  0.9266213
##  [4,] -0.3884971  0.1115714  0.03235299 -0.01829845 -0.4227808
##  [5,]  0.5681719  0.6977379  2.29922078 -0.64166100 -2.3917737
##  [6,]  0.7653842  2.1675478 -1.72970175  0.36498124  0.3402267
```

```
##  [7,] -0.5500326  0.8666017  1.81512428  2.11950376  0.1361102
##  [8,] -0.4797463  0.7774203 -0.78021109 -0.56915313  0.4102409
##  [9,] -0.1471528  0.7052118  0.91228837  0.15053963  0.6806240
## [10,]  0.2748029  0.5175826  1.69657567  0.63888095 -1.1618406
```

```r
abs((L_est%*%t(R_est)-M_true)/M_true)
```

```
##               [,1]         [,2]         [,3]         [,4]         [,5]
##  [1,] 2.018975e-07 1.051809e-07 9.743451e-08 1.472509e-06 1.144619e-07
##  [2,] 2.094439e-08 1.075560e-06 1.607039e-07 4.638386e-08 4.525460e-07
##  [3,] 2.063015e-08 7.712183e-09 9.271611e-09 7.584682e-09 1.091071e-06
##  [4,] 1.537262e-07 3.774051e-08 1.023559e-08 5.103393e-08 1.132821e-06
##  [5,] 4.406606e-08 5.329545e-08 1.435542e-08 3.403746e-08 1.412819e-07
##  [6,] 1.372068e-07 1.053674e-08 6.922238e-09 9.912175e-07 6.447396e-08
##  [7,] 7.938317e-06 9.245591e-08 6.455912e-08 3.348250e-07 1.254775e-06
##  [8,] 6.032951e-08 2.068664e-08 1.326588e-07 3.056847e-08 5.169969e-06
##  [9,] 7.578955e-08 6.414347e-07 7.957414e-06 2.961776e-07 1.103768e-06
## [10,] 9.186841e-07 1.829936e-07 5.765604e-08 4.743181e-07 7.273638e-07
##               [,6]         [,7]         [,8]         [,9]        [,10]
##  [1,] 1.304628e-06 1.519419e-08 2.992847e-07 2.276652e-07 2.187569e-08
##  [2,] 9.277565e-08 4.266561e-08 3.913180e-07 6.112209e-07 1.294174e-07
##  [3,] 2.093487e-08 3.409272e-09 5.347970e-08 4.854896e-07 3.456530e-08
##  [4,] 9.819064e-08 4.926626e-08 8.960945e-08 8.670803e-07 5.094039e-08
##  [5,] 1.694172e-08 9.350785e-09 1.655079e-07 1.057372e-07 7.803566e-08
##  [6,] 2.373333e-08 1.927381e-07 8.946691e-09 5.347248e-08 7.074760e-09
##  [7,] 2.304779e-08 1.171808e-06 3.889999e-06 2.127470e-07 8.983849e-08
##  [8,] 5.569547e-08 7.194797e-08 2.100071e-08 7.307222e-08 3.890612e-08
##  [9,] 3.049756e-07 2.535980e-08 8.870243e-08 1.198197e-06 2.441435e-07
## [10,] 2.329388e-08 1.877795e-08 1.133234e-07 8.197081e-07 1.502970e-07
```

If we increase the number of missing values to 30%, GD is not able to recover the true matrix. The relative loss $\frac{\|\mathcal{P}_\Omega(LR^T-M_{obs})\|_F}{\|M_{obs}\|_F}$ and $\frac{\|(LR^T-M^*)\|_F}{\|M^*\|_F}$ are as following

```
## [1] 0.002295485
```

```
## [1] 0.5491497
```

Regularized GD

Regularized objective function is

$$
\begin{aligned}
f_{reg}(L,R) &= \frac{1}{2}\|\mathcal{P}_\Omega(LR^T-M)\|_F^2 + G(L,R), \\
G(L,R) &= \rho\sum_{i=1}^{n_1} G_0\Big(\frac{3\|L^{(i)}\|^2}{2\beta_1^2}\Big) + \rho\sum_{j=1}^{n_2} G_0\Big(\frac{3\|R^{(j)}\|^2}{2\beta_2^2}\Big) + \rho\; G_0\Big(\frac{3\|L\|_F^2}{2\beta_T^2}\Big) + \rho\; G_0\Big(\frac{3\|R\|_F^2}{2\beta_T^2}\Big),
\end{aligned}
$$

where

$$G_0(z) = \mathbf{1}(z > 1)(z-1)^2,$$
$$\beta_T = \sqrt{C_T r \Sigma_{max}}, \Sigma_{max} \text{ is the maximum singular value of } M^*,$$
$$\beta_1 = \beta_T \sqrt{\frac{3\mu r}{n_1}}, \text{(suppose } M^* \text{ is } \mu\text{-incoherent)},$$
$$\beta_2 = \beta_T \sqrt{\frac{3\mu r}{n_2}},$$
$$\rho = 8p\delta_0^2, \delta_0 = \frac{\delta}{6}, \delta = \frac{\Sigma_{min}}{C_d r^{1.5}\kappa}, \kappa = \frac{\Sigma_{max}}{\Sigma_{min}}.$$

Where C_T, C_d are absolute constants. Since M^* is unknown, it is suggested to estimate Σ_{max} and μ by

$$\Sigma_{max} \approx C_1 \sqrt{\frac{\|\mathcal{P}_\Omega(M)\|_F^2}{p^r}}, \qquad \mu \approx C_2 \frac{\sqrt{n_1 n_2}}{r\Sigma_{max}} \max_{(i,j)\in\Omega} |M_{ij}|,$$

where C_1, C_2 are absolute constants.

Row-scaled Spectral Initialization

- Obtain L_0, R_0 by SVD (same as vanilla GD).
- Scale the rows of L_0 and R_0 to make them incoherent (i.e. bounded row-norm), that is

$$\|L_0\|_{2,\infty} \le \sqrt{\frac{2}{3}}\beta_1$$

$$\|R_0\|_{2,\infty} \le \sqrt{\frac{2}{3}}\beta_2$$

```
# Regularized GD -----------------------------------------------------

RowScalSpecInit = function(M, r, beta1, beta2, p) ## beta1,2: target row norm bounds. p: observe probab
{
  rsvd = rsvd(M/p, r)
  D = diag(rsvd$d); U = rsvd$u; V = rsvd$v #### Note: assume r is given.
  L0 = U %*% sqrt(D)
  R0 = V %*% sqrt(D)
  for (i in 1:dim(M)[1]) {
    if (norm(L0[i,], '2') > sqrt(2/3)*beta1) {
      L0[i,] = L0[i,]/norm(L0[i,], '2')*sqrt(2/3)*beta1
    }
  }
  for (i in 1:dim(M)[2]) {
    if (norm(R0[i,], '2') > sqrt(2/3)*beta2) {
      R0[i,] = R0[i,]/norm(R0[i,], '2')*sqrt(2/3)*beta2
    }
  }
  return(list(L0=L0, R0=R0))
}


G0Derivative = function(z) I(z>1)*2*(z-1)
```

```r
ReguGD = function(M, L0, R0, rho, beta1, beta2, betaT, lr, MaxIter){
  Lt = L0; Rt = R0;
  for (i in 1:MaxIter) {
    L = Lt - lr * (Projector(Lt%*%t(Rt)-M)%*%Rt +
                     rho*(3/beta1^2*Lt)*G0Derivative(3/(2*beta1^2)*apply(Lt, 1, function(x) norm(x, '2'
                     rho*(3/betaT^2*Lt)*G0Derivative(3/(2*betaT^2)*norm(Lt, 'f')^2))
    R = Rt - lr * (t(Projector(Lt%*%t(Rt)-M))%*%Lt +
                     rho*(3/beta2^2*Rt)*G0Derivative(3/(2*beta2^2)*apply(Rt, 1, function(x) norm(x, '2'
                     rho*(3/betaT^2*Rt)*G0Derivative(3/(2*betaT^2)*norm(Rt, 'f')^2))
    Lt = L
    Rt = R
  }
  return(list(Lt=Lt, Rt=Rt))
}

Ct = 5
Cd = 6500
C1 = 0.4
C2 = 1
MaxSingVal = max(rsvd(M_obs/(1-missfrac), r)$d) ### = C1 * sqrt(norm(M_obs, 'f')^2/(1-missfrac)^r)
mu = C2 * sqrt(n1*n2) / (r*MaxSingVal) * max(abs(M_obs))
MinSingVal = min(rsvd(M_obs/(1-missfrac), r)$d)
kappa = MaxSingVal / MinSingVal
delta = MinSingVal/(Cd*r^1.5*kappa)
delta0 = delta / 6
betaT = sqrt(Ct*r*MaxSingVal)
beta1 = betaT * sqrt(3*mu*r/n1)
beta2 = betaT * sqrt(3*mu*r/n2)
rho = 8 * (1-missfrac) * delta0^2


LR_init =  RowScalSpecInit(M_obs, r, beta1, beta2, 1-missfrac)
L0 = LR_init$L0; R0 = LR_init$R0
LR_est_regu = ReguGD(M_obs, L0, R0, rho, beta1, beta2, betaT, 0.01, 10000)
L_est_reg = LR_est_regu$Lt; R_est_reg = LR_est_regu$Rt

norm(Projector(L_est_reg%*%t(R_est_reg)-M_obs), 'f')/norm(M_obs, 'f')
```

```
## [1] 8.046773e-08
```

```r
norm(L_est_reg%*%t(R_est_reg)-M_true, 'f')/norm(M_true, 'f')
```

```
## [1] 6.672459e-07
```

Projected GD

The key part is to project L and R onto the set of incoherent matrices in each step, that is,

$$L_{t+1} = \mathcal{P}_{\mathcal{C}}(L_t - \eta \nabla_L f(L_t, R_t)),$$
$$R_{t+1} = \mathcal{P}_{\mathcal{C}}(R_t - \eta \nabla_R f(L_t, R_t)),$$

where

$$\mathcal{C} = \left\{ X \in \mathbb{R}^{n\times r} \middle| \|X\|_{2,\infty} \leq \sqrt{\frac{c\mu r}{n}} \|X_0\| \right\}.$$

```r
# Projected GD ------------------------------------------------------------

IncohProj = function(X, X0, c, mu){
  n = dim(X)[1]; r  = dim(X)[2]
  for (i in 1:n) {
    if (norm(X[i,], '2') > sqrt(c*mu*r/n)*norm(X0, '2')) {
      X[i,] = X[i,] / norm(X[i,], '2') * sqrt(c*mu*r/n) * norm(X0, '2')
    }
  }
  return(X)
}


ProjectedGD = function(M, L0, R0, c, mu, lr, MaxIter)
{
  Lt = L0; Rt = R0;
  for (t in 1:MaxIter) {
    L = Lt - lr * Projector(Lt%*%t(Rt)-M) %*% Rt
    L = IncohProj(L, L0, c, mu)
    R = Rt - lr * t(Projector(Lt%*%t(Rt)-M)) %*% Lt
    R = IncohProj(R, R0, c, mu)
    Lt = L
    Rt = R
  }
  return(list(Lt = Lt, Rt = Rt))
}


c = 2
mu = .5
lr = 0.01
MaxIter = 10000


L0 = LR_decomp(M_obs/(1-missfrac))$L0; R0 = LR_decomp(M_obs/(1-missfrac))$R0
LR_est_proj = ProjectedGD(M_obs, L0, R0, c, mu, lr, MaxIter)
L_est_proj = LR_est_proj$Lt; R_est_proj = LR_est_proj$Rt

norm(Projector(L_est_proj%*%t(R_est_proj)-M_obs), 'f')/norm(M_obs, 'f')
```

```
## [1] 8.176426e-08
```

```r
norm((L_est_proj%*%t(R_est_proj)-M_true), 'f')/norm(M_true, 'f')
```

```
## [1] 6.780138e-07
```

GD on Manifold

Unlike previous methods, here we try to use SVD to estimate the complete matrix. That is, minimize the loss function as follows

$$\operatorname{minimize} F(L, R) \qquad \text{s.t.} \quad L \in \mathcal{G}\left(n_1, r\right), R \in \mathcal{G}(n_2, r),$$

where

$$F(\boldsymbol{L}) := \min_{\boldsymbol{R} \in \mathbb{R}_2^n \times r} \left\| \mathcal{P}_\Omega \left(\boldsymbol{M}^* - \boldsymbol{L} \boldsymbol{R}^\top \right) \right\|_\mathrm{F}^2$$

$$X^T \mathcal{P}_\Omega(XSY^T)Y = X^T \mathcal{P}_\Omega(M)Y$$

$$L^T \mathcal{P}_\Omega(M) = L^T \mathcal{P}_\Omega(LR^T)$$

Initialization

AltMin

ADMM