1 A Toy Example

In this section, we define a simple objective function called *eval*() which calculates the sum of a penalty term and the squared error between the DataRemix reconstruction and the original input matrix. This example illustrates how to include additional parameters which are necessary for the customized evaluation function fn() into DataRemix framework. The input matrix is a 100-by-9 matrix with random values. In this case, we know that when k=9,p=1 or $\mu = 1$, p=1, DataRemix reconstruction is the same as the orginial matrix and the objective function achieves the minimal value which is qual to the penalty term we add.

```
> library(DataRemix)
> eval <- function(X_reconstruct, X, penalty){
+   return(-sum((X-X_reconstruct)^2)+penalty)
+ }#eval
```

First we genrate a random matrix with dimension 100-by-9 and perform the SVD decomposition.

```
> set.seed(1)
> num_of_row <- 100
> num_of_col <- 9
> X <- matrix(rnorm(num_of_row*num_of_col), nrow = num_of_row, ncol = num_of_col)
> svdres <- svd(X)
```

Set mt to be 2000.

```
> basis_short <- omega[1:2000,]
```

Infer the optimal combinations of k, p and μ. Here X and *penalty* are additional inputs for the *eval*() function. If we have the full SVD decomposition, we can leave matrix as NULL. For some large-scale matrices, if the SVD computation is time intensive, we don't need to finish the full SVD. Instead we can just compute the SVD decomposition up to a sufficient rank and inlcude the original gene expression profile to calculate the residual.

```
> DataRemix.res <- DataRemix(svdres, matrix = NULL, eval,
+                  k_limits = c(1, length(svdres$d)), p_limits = c(-1,1),
+                  mu_limits = c(1e-12,1), num_of_initialization = 5,
+                  num_of_thompson = 50, basis = basis_short, xi = 0.1,
+                  full = T, verbose = F, X = X, penalty = 100)
> knitr::kable(cbind(1:55,DataRemix.res$para), align = "l",
+              col.names = c("Iteration","k","p","mu","Eval"))
```

Iteration	k	p	mu	Eval
1	8	0.9343941	0.8669163	80.1334695
2	4	-0.6161244	0.0822944	-774.5493431
3	6	-0.8592770	0.5276627	-674.5081307

4	5	-0.9036173	0.5945408	-595.2096802
5	8	0.1977374	0.0279159	-608.4540773
6	5	-0.3638044	0.0000241	-813.6702821
7	2	-0.8046778	0.0000000	-853.4656232
8	9	1.0000000	0.0002110	100.0000000
9	9	0.6661921	0.0154153	-184.7067747
10	8	0.6531390	0.0000000	-244.3300655
11	8	1.0000000	0.2141172	62.4173374
12	9	1.0000000	0.0000423	100.0000000
13	1	0.8418133	0.0000000	-717.5578470
14	9	1.0000000	0.5691544	100.0000000
15	9	0.6548131	1.0000000	-197.4167734
16	4	1.0000000	0.0000000	-312.2857721
17	9	0.8373235	0.0000000	2.5059486
18	7	1.0000000	0.0000000	-36.8397906
19	9	0.9417058	0.0015476	84.1358492
20	5	1.0000000	1.0000000	100.0000000
21	8	1.0000000	0.0000000	39.1484035
22	8	1.0000000	0.0900272	49.6118042
23	3	0.9365716	0.0889231	-349.7091651
24	6	1.0000000	0.9720235	99.8300098
25	8	0.9251656	0.0003053	15.2575305
26	9	1.0000000	0.0211273	100.0000000
27	5	0.8231541	0.0000024	-282.8900039
28	8	1.0000000	0.0024223	39.4428521
29	9	-0.0882469	0.0000000	-716.3423732
30	9	1.0000000	0.0000002	100.0000000
31	5	1.0000000	0.0749025	-158.2670247
32	3	0.7020952	0.3133165	-267.6340442
33	8	0.9529303	0.0001027	29.0614097
34	5	0.7020183	1.0000000	-76.1587541
35	1	0.1021742	1.0000000	-28.7859003
36	5	0.8742549	0.5679676	-2.4428053
37	7	0.7783341	0.0000000	-176.7300645
38	7	0.9761727	0.0000957	-39.3551455
39	9	-0.7851174	0.0000025	-830.6108355
40	9	0.6304672	0.0000001	-224.3579299
41	1	0.7011276	0.9561123	53.2877702
42	7	0.7868272	0.0000064	-168.6494520
43	8	1.0000000	0.0000009	39.1485116
44	8	1.0000000	0.4135032	79.0683587
45	6	0.8345570	0.0000000	-198.0291845
46	1	0.0284351	0.0000001	-833.9386866
47	8	0.9008744	0.0000000	-0.5474068
48	9	1.0000000	0.0000000	100.0000000
49	2	0.9689157	0.1544205	-370.4069143

```
|50        |4  |0.9904412  |1.0000000 |99.7032505   |
|51        |9  |1.0000000  |0.0000000 |100.0000000  |
|52        |9  |1.0000000  |0.0234981 |100.0000000  |
|53        |7  |1.0000000  |1.0000000 |100.0000000  |
|54        |7  |0.3135010  |0.0025860 |-571.7045348 |
|55        |7  |-0.9857505 |0.0000000 |-845.5505885 |
```

2 GTex Correlation Network

In this section, we define a different task of optimizing the known pathway recovery based on the GTex gene expression data. $corMatToAUC()$ is the main objective function with two inputs: *data* and GS. We formally define the objective as the average AUC across pathways and we also keep track of the average AUPR value. You can refer to the $corMatToAUC()$ document for more information.

```
> library(DataRemix)
```

Load the data. $GTex\_cc$ stands for the GTex gene correlation matrix with dimension 7294-by-7294 and *canonical* represents the canonical mSigDB pathways with dimension 7294-by-1330. It takes time to decompose $GTex\_cc$, thus we pre-compute the SVD decomposition of $GTex\_cc$ and load it as $GTex\_svdres$.

```
> load(url("https://www.dropbox.com/s/o949wkg76k0ccaw/GTex_cc.rdata?dl=1"))
> load(url("https://www.dropbox.com/s/wsuze8w2rp0syqg/GTex_svdres.rdata?dl=1"))
> load(url("https://github.com/wgmao/DataRemix/blob/master/inst/extdata/canonical.rdata?raw=
> #svdres <- svd(GTex_cc)
```

Run corMatToAUC() on the default correlation matrix $GTex\_cc$.

```
> GTex_default <- corMatToAUC(GTex_cc, canonical)
> GTex_default
```

```
[1] 0.0450869 0.7238648
```

Set mt to be 2000.

```
> basis_short <- omega[1:2000,]
```

Infer the optimal combinations of k, p and μ. Here GS is the additional input for the $corMatToAUC()$ function.

```
> DataRemix.res <- DataRemix(GTex_svdres, GTex_cc,corMatToAUC,
+                            k_limits = c(1, length(GTex_svdres$d)%/%2),
+                            p_limits = c(-1,1), mu_limits = c(1e-12,1),
+                            num_of_initialization = 5, num_of_thompson = 150,
+                            basis = basis_short, xi = 0.1, full = T, verbose = F,
+                            GS = canonical)
> knitr::kable(cbind(1:15,DataRemix.res$full[order(DataRemix.res$para[,4],
+             decreasing = T)[1:15],]), align = "l", col.names = c("Rank",
+             "k","p","mu","mean AUPR", "mean AUC"))
```

Rank	k	p	mu	mean AUPR	mean AUC
1	1175	0.3094122	0.0000001	0.1037430	0.7758704
2	2193	0.3224210	0.0007957	0.1067562	0.7758402
3	674	0.3092313	0.0000000	0.0992667	0.7755660
4	1960	0.3387817	0.0000004	0.1046918	0.7755577
5	1649	0.3305281	0.0000022	0.1047632	0.7755438
6	1201	0.2953043	0.0046469	0.1050208	0.7755354
7	1822	0.3051588	0.0029203	0.1072695	0.7754740
8	826	0.2727391	0.1386776	0.1036309	0.7754550
9	2781	0.3299403	0.0002011	0.1063461	0.7753907
10	2036	0.3529152	0.1020208	0.1037271	0.7753318
11	3269	0.3464010	0.0000000	0.1047787	0.7752678
12	1476	0.3265687	0.0000000	0.1041608	0.7752523
13	3097	0.3212749	0.0000000	0.1070279	0.7752323
14	1433	0.3581519	1.0000000	0.1018863	0.7752311
15	3335	0.3501867	0.0118857	0.1044306	0.7752071