Eigenvectors from Eigenvalues Sparse Principal Component Analysis (EESPCA) R logic for simple example results

H. Robert Frost *

The following R logic was used to generate the results associated with the running example in the EESPCA manuscript and is included as a vignette in the EESPCA R package. This example is based on a data set simulated according to a 10-dimensional multivariate normal (MVN) distribution.

0.1 Load required R packages

The MASS package is used to simulate the MVN data, the PMA package contains the implementation of the Witten et al. LASSO-based sparse principal components method [2], the rifle package contains the implementation the rifle method by Tan et al. [1], and the EESPCA package contains the implementation of the EESPCA method and TPower method by Yuan and Zhang [3] as well as logic to support sparsity parameter selection for both TPower and rifle via cross-validation.

```
> library(MASS)
> library(PMA)
> library(rifle)
> library(EESPCA)
```

0.2 Set random seed

Set the random seed so that simulation is reproducible.

```
> set.seed(2)
```

0.3 Set parameters that control the simulation

```
> p =10 # number of variables
> prop.info = 0.4 # proportion of variables that have non-zero loadings on the 1st PC
> rho = 0.5 # covariance between informative variables
> n = 100 # simulated sample size
```

*rob.frost@dartmouth.edu, Department of Biomedical Data Science, Geisel School of Medicine, Dartmouth College, Hanover, NH 03755

0.4 Define MVN distribution

The population mean for data is set to the zero vector and the population covariance matrix is given a block covariance structure with one block including the first 4 variables and one block including the last two variables. To simplify the example, we set the population variance for all variables to 1, which aligns with the common practice of performing PCA after standardization.

```
> S = matrix(0, nrow=p, ncol=p)
> num.info = p*prop.info
> S[1:num.info, 1:num.info] = rho
> S[p,p-1] = S[p-1,p] = rho
> diag(S) = 1
> S
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]  1.0  0.5  0.5  0.5    0    0    0    0  0.0   0.0
 [2,]  0.5  1.0  0.5  0.5    0    0    0    0  0.0   0.0
 [3,]  0.5  0.5  1.0  0.5    0    0    0    0  0.0   0.0
 [4,]  0.5  0.5  0.5  1.0    0    0    0    0  0.0   0.0
 [5,]  0.0  0.0  0.0  0.0    1    0    0    0  0.0   0.0
 [6,]  0.0  0.0  0.0  0.0    0    1    0    0  0.0   0.0
 [7,]  0.0  0.0  0.0  0.0    0    0    1    0  0.0   0.0
 [8,]  0.0  0.0  0.0  0.0    0    0    0    1  0.0   0.0
 [9,]  0.0  0.0  0.0  0.0    0    0    0    0  1.0   0.5
[10,]  0.0  0.0  0.0  0.0    0    0    0    0  0.5   1.0
```

0.5 Compute population PCs

For this population covariance matrix, the first population PC has equal non-zero loadings (-0.5) for just the first four variables and the second population PC has equal non-zero loadings (0.7071068) for just the last two variables. The variances of these population PCs are 2.5 and 1.5.

```
> (eigen.out = eigen(S))
```

```
eigen() decomposition
$values
 [1] 2.5 1.5 1.0 1.0 1.0 1.0 0.5 0.5 0.5 0.5

$vectors
      [,1]      [,2] [,3] [,4] [,5] [,6]       [,7]       [,8]       [,9]
 [1,] -0.5 0.0000000    0    0    0    0  0.0000000  0.8660254  0.0000000
 [2,] -0.5 0.0000000    0    0    0    0  0.0000000 -0.2886751 -0.5773503
 [3,] -0.5 0.0000000    0    0    0    0  0.0000000 -0.2886751 -0.2113249
 [4,] -0.5 0.0000000    0    0    0    0  0.0000000 -0.2886751  0.7886751
```

```
 [5,]  0.0 0.0000000    0    0    0    1  0.0000000  0.0000000  0.0000000
 [6,]  0.0 0.0000000    0    0    1    0  0.0000000  0.0000000  0.0000000
 [7,]  0.0 0.0000000    0    1    0    0  0.0000000  0.0000000  0.0000000
 [8,]  0.0 0.0000000    1    0    0    0  0.0000000  0.0000000  0.0000000
 [9,]  0.0 0.7071068    0    0    0    0  0.7071068  0.0000000  0.0000000
[10,]  0.0 0.7071068    0    0    0    0 -0.7071068  0.0000000  0.0000000
           [,10]
 [1,]  0.0000000
 [2,] -0.5773503
 [3,]  0.7886751
 [4,] -0.2113249
 [5,]  0.0000000
 [6,]  0.0000000
 [7,]  0.0000000
 [8,]  0.0000000
 [9,]  0.0000000
[10,]  0.0000000
```

0.6 Compute approximate normed squared loadings for the first population PC.

The approximate normed squared loadings for the first population PC according the formula used by the EESPCA method are:

```
> v1 = eigen.out$vectors[,1]
> lambda1 = eigen.out$values[1]
> (approx.v1.sq = computeApproxNormSquaredEigenvector(S, v1, lambda1, trace=T))

 [1] 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.0 0.0 0.0
```

0.7 Simulated MVN data

Simulate one set of 100 independent samples are drawn from this MVN distribution and compute the sample covariance matrix.

```
> X = mvrnorm(n=n, mu=rep(0,p), Sigma=S)
> S.hat = cov(X)
```

0.8 Perform standard PCA

```
> prcomp.out = prcomp(X)
> (V.2 = prcomp.out$rotation[,1:2])

               PC1          PC2
 [1,] -0.540731693  0.113426021
```

```
 [2,] -0.445638863 -0.085275311
 [3,] -0.505875932 -0.055269937
 [4,] -0.496005996 -0.126424305
 [5,] -0.032694438  0.156095617
 [6,]  0.021685149 -0.048132883
 [7,]  0.000871458  0.002178131
 [8,] -0.022737959 -0.220974342
 [9,] -0.048711523  0.724404926
[10,] -0.051508009  0.600454221
```

```
> prcomp(X)$sdev[1:2]^2
```

```
[1] 3.392936 1.521292
```

0.9 Compute PCA reconstruction error

Compute the rank 2 reconstruction based on the first two PCs (computed as the squared Frobenius norm of the residual matrix). By definition, this is the minimum rank 2 reconstruction error. It is important to note that this reconstruction error is distinct from the out-of-sample reconstruction error used for comparative analysis in the EESPCA paper and used as the objective function for cross-validation based selection of sparsity parameters.

```
> (pca.error = reconstructionError(X, V.2))
```

```
[1] 574.8416
```

Calculate the Euclidean distance between the true PC loadings and the estimated loadings for the first PC.

```
> computeEuclideanDistance = function(x1, x2) {
+    # To account for the arbitrary sign of eigenvectors, taking absolute value first.
+   return (sqrt(sum((abs(x1) - abs(x2))^2)))
+ }
> (pca.L2 = computeEuclideanDistance(v1, prcomp.out$rotation[,1]))
```

```
[1] 0.108386
```

0.10 Compute approximate normed squared loadings for the first sample PC.

```
> v1 = prcomp.out$rotation[,1]
> v1^2
```

```
 [1] 2.923908e-01 1.985940e-01 2.559105e-01 2.460219e-01 1.068926e-03
 [6] 4.702457e-04 7.594391e-07 5.170148e-04 2.372812e-03 2.653075e-03
```

```
> lambda1 = prcomp.out$sdev[1]^2
> (approx.v1.sq = computeApproxNormSquaredEigenvector(S.hat, v1, lambda1, trace=F))

 [1] 2.467070e-01 1.745954e-01 2.112992e-01 2.037578e-01 7.489176e-04
 [6] 3.311500e-04 5.704496e-07 3.445289e-04 1.574658e-03 1.873035e-03

> (ratio = sqrt(approx.v1.sq)/abs(v1))

 [1] 0.9185629 0.9376340 0.9086673 0.9100603 0.8370341 0.8391701 0.8666868
 [8] 0.8163217 0.8146318 0.8402300
```

0.11 Compute the first two sparse PCs using EESPCA method

```
> eespca.out = eespcaForK(X, k=2, trace=F, compute.sparse.lambda=T)
> eespca.out

$V
           PC_1      PC_2
 [1,] 0.5431183 0.0000000
 [2,] 0.4568986 0.0000000
 [3,] 0.5026347 0.0000000
 [4,] 0.4935834 0.0000000
 [5,] 0.0000000 0.0000000
 [6,] 0.0000000 0.0000000
 [7,] 0.0000000 0.0000000
 [8,] 0.0000000 0.0000000
 [9,] 0.0000000 0.7790692
[10,] 0.0000000 0.6269379

$lambdas
[1] 3.377988 1.460073
```

Compute the rank 2 reconstruction error using the EESPCA sparse PCs:

```
> (eespca.error = reconstructionError(X, eespca.out$V))

[1] 582.3821
```

Calculate the L2 distance between the true PC loadings and the estimated sparse loadings for the first PC.

```
> (eespca.L2 = computeEuclideanDistance(v1, eespca.out$V[,1]))

[1] 0.08503932
```

0.12 Compute the first two sparse PCs using EESPCA.cv method

Find optimal penalty via 5-fold cross-validation for first PC:

```
> p = ncol(X)
> default.thresh = 1/sqrt(p)
> sparse.threshold.values=seq(from=0.75*default.thresh, to=1.25*default.thresh,
+         length.out=21)
> cv.out = eespcaCV(X, sparse.threshold.values=sparse.threshold.values)
```

Compute sparse PC 1 using penalty that generated minimum error on fist PC

```
> eespca.cv.v1 = eespca(X, sparse.threshold=cv.out$best.sparsity, compute.sparse.lambd
```

Find optimal penalty for second PC using cross-validation on residual matrix. Use that to compute sparse PC 2.

```
> X.resid = computeResidualMatrix(X, eespca.cv.v1)
> cv.out = eespcaCV(X.resid, sparse.threshold.values=sparse.threshold.values)
> eespca.cv.v2 =  eespca(X.resid, sparse.threshold=cv.out$best.sparsity, compute.spars
> (V = cbind(eespca.cv.v1,eespca.cv.v2))
```

```
      eespca.cv.v1 eespca.cv.v2
 [1,]    0.5431183    0.0000000
 [2,]    0.4568986    0.0000000
 [3,]    0.5026347    0.0000000
 [4,]    0.4935834    0.0000000
 [5,]    0.0000000    0.0000000
 [6,]    0.0000000    0.0000000
 [7,]    0.0000000    0.0000000
 [8,]    0.0000000    0.0000000
 [9,]    0.0000000    0.7790692
[10,]    0.0000000    0.6269379
```

Compute the rank 2 reconstruction error using the EESPCA.cv sparse PCs:

```
> (eespca.cv.error = reconstructionError(X, V))
```

```
[1] 582.3821
```

Calculate the L2 distance between the true PC loadings and the estimated sparse loadings for the first PC.

```
> (eespca.cv.L2 = computeEuclideanDistance(v1, eespca.cv.v1))
```

```
[1] 0.08503932
```

0.13 Use the Witten et al. method [2] to compute the first two sparse PCs

Find optimal penalty via 5-fold cross-validation for first PC:

```
> cv.out =  SPC.cv(X, sumabsv=seq(1, sqrt(p), len=20), niter=10, trace=F)
```

Look at solution at penalty that generated minimum error on fist PC (this applies the same penalty to both the first and second PCs and does not enforce orthogonality):

```
> spc.out=SPC(X, sumabsv=cv.out$bestsumabsv, K=2, trace=F)
> spc.out$v

             [,1]        [,2]
 [1,] -0.54059796  0.13111568
 [2,] -0.44564166 -0.05782282
 [3,] -0.50865986 -0.02326875
 [4,] -0.49999193 -0.09621028
 [5,]  0.00000000  0.14463543
 [6,]  0.00000000 -0.03042624
 [7,]  0.00000000  0.00000000
 [8,]  0.00000000 -0.19873804
 [9,] -0.01472116  0.73352834
[10,] -0.01462424  0.60849121

> (spc.error = reconstructionError(X, spc.out$v))

[1] 575.4584
```

Look at solution at penalty 1 SE from value that generated minimum error:

```
> spc.1se.out =SPC(X, sumabsv=cv.out$bestsumabsv1se, K=2, trace=F)
> spc.1se.out$v

            [,1]         [,2]
 [1,] -0.7432266  0.068410686
 [2,] -0.1235048 -0.013146635
 [3,] -0.4631567 -0.002846501
 [4,] -0.4667404 -0.086864933
 [5,]  0.0000000  0.120546022
 [6,]  0.0000000  0.000000000
 [7,]  0.0000000  0.000000000
 [8,]  0.0000000 -0.126978432
 [9,]  0.0000000  0.751060194
[10,]  0.0000000  0.626775275

> (spc.1se.error = reconstructionError(X, spc.1se.out$v))
```

```
[1] 620.0697
```

Calculate the L2 distance between the true PC loadings and the estimated sparse loadings for the first PC.

```
> (spc.L2 = computeEuclideanDistance(v1, spc.out$v[,1]))
```

```
[1] 0.06779645
```

```
> (spc.1se.L2 = computeEuclideanDistance(v1, spc.1se.out$v[,1]))
```

```
[1] 0.3931142
```

0.14 Use the Yuan and Zhang TPower method [3] to compute the first two sparse PCs

Find optimal k-value via 5-fold cross-validation for first PC:

```
> k.values = round(seq(1, p, len=20))
> (optimal.k = tpowerPCACV(X=X, k.values=k.values, nfolds=5))
```

```
[1] 4
```

Generate initial PC loadings using non-truncated powerIteration:

```
> v.init = powerIteration(X=S.hat)$v1
```

Generate first and second sparse PC loadings using TPower method (second sparse PC is generated on residual matrix using new optimal sparsity value):

```
> tpower.v1 = tpower(X=S.hat, max.iter=100, k=optimal.k, v1.init=v.init)
> X.resid = computeResidualMatrix(X, tpower.v1)
> (optimal.k2 = tpowerPCACV(X=X.resid, k.values=k.values, nfolds=5))
```

```
[1] 2
```

```
> X.resid.cov = cov(X.resid)
> v2.init = powerIteration(X=X.resid.cov)$v1
> tpower.v2 = tpower(X=X.resid.cov, max.iter=100, k=optimal.k2, v1.init=v2.init)
> (v = cbind(tpower.v1, tpower.v2))
```

```
      tpower.v1 tpower.v2
 [1,] 0.5408860 0.0000000
 [2,] 0.4479383 0.0000000
 [3,] 0.5070190 0.0000000
 [4,] 0.4997253 0.0000000
 [5,] 0.0000000 0.0000000
 [6,] 0.0000000 0.0000000
 [7,] 0.0000000 0.0000000
 [8,] 0.0000000 0.0000000
 [9,] 0.0000000 0.7556217
[10,] 0.0000000 0.6550083
```

```
> (tpower.error = reconstructionError(X, v))
```

```
[1] 582.2063
```

Calculate the L2 distance between the true PC loadings and the estimated sparse loadings for the first PC.

```
> (tpower.L2 = computeEuclideanDistance(v1, tpower.v1))
```

```
[1] 0.08428099
```

0.15 Use the Tan et al. rifle method [1] to compute the first two sparse PCs

Find optimal k-value via 5-fold cross-validation for first PC:

```
> k.values = round(seq(1, p, len=20))
> (optimal.k = riflePCACV(X=X, k.values=k.values, nfolds=5))
```

```
[1] 4
```

Generate initial PC loadings using recommended rifle approach (as implemented in rifle.init() method):

```
> v.init = rifleInit(X)
```

Generate first and second sparse PC loadings using rifle method (second sparse PC is generated on residual matrix using new optimal sparsity value):

```
> rifle.v1 = rifle(A=S.hat, B=diag(p), init=v.init, k=optimal.k)
> X.resid = computeResidualMatrix(X, rifle.v1)
> (optimal.k2 = riflePCACV(X=X.resid, k.values=k.values, nfolds=5))
```

```
[1] 2
```

```
> X.resid.cov = cov(X.resid)
> v2.init = rifleInit(X.resid)
> rifle.v2 = rifle(A=X.resid.cov, B=diag(p), init=v2.init, k=optimal.k2)
> (v = cbind(rifle.v1, rifle.v2))
```

```
           [,1]      [,2]
 [1,] 0.5507995 0.0000000
 [2,] 0.4347170 0.0000000
 [3,] 0.5081865 0.0000000
 [4,] 0.4993871 0.0000000
 [5,] 0.0000000 0.0000000
 [6,] 0.0000000 0.0000000
 [7,] 0.0000000 0.0000000
 [8,] 0.0000000 0.0000000
 [9,] 0.0000000 0.7747091
[10,] 0.0000000 0.6323178
```

```
> (rifle.error = reconstructionError(X, v))
```

```
[1] 582.377
```

Calculate the L2 distance between the true PC loadings and the estimated sparse loadings for the first PC.

```
> (rifle.L2 = computeEuclideanDistance(v1, -rifle.v1))
```

```
[1] 0.08555848
```

0.16 Summary of reconstruction errors and estimation distances

```
> estimation.summary = data.frame(pca=c(pca.error, pca.L2),
+         spc=c(spc.error, spc.L2),
+         spc.1se=c(spc.1se.error, spc.1se.L2),
+         eespca=c(eespca.error, eespca.L2),
+         tpower=c(tpower.error, tpower.L2),
+         rifle=c(rifle.error, rifle.L2))
> rownames(estimation.summary) = c("Reconstruction Error", "Loadings distance")
> estimation.summary
```

```
                            pca          spc     spc.1se       eespca
Reconstruction Error 574.841592 575.45837865 620.0696917 582.38209546
Loadings distance      0.108386   0.06779645   0.3931142   0.08503932
                           tpower        rifle
Reconstruction Error 582.20628324 582.37695461
Loadings distance      0.08428099   0.08555848
```

References

[1] Kean Ming Tan, Zhaoran Wang, Han Liu, and Tong Zhang. Sparse generalized eigenvalue problem: optimal statistical rates via truncated rayleigh flow. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 80(5):1057–1086, 2018.

[2] Daniela M. Witten, Robert Tibshirani, and Trevor Hastie. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, 10(3):515–534, July 2009.

[3] Xiao-Tong Yuan and Tong Zhang. Truncated power method for sparse eigenvalue problems. *J. Mach. Learn. Res.*, 14(1):899–925, April 2013.