PCA, SVD and Mahalanobis distance

Christopher Gillies

11/16/2017

Multivariate normal distribution

$$f(x) = \frac{1}{\sqrt{(2\pi)^n|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(x-\mu)^T\boldsymbol{\Sigma}^{-1}(x-\mu)\right) \quad (1)$$

If we assume the data are zero-centered then:

$$f(x) = \frac{1}{\sqrt{(2\pi)^n|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(x)^T\boldsymbol{\Sigma}^{-1}(x)\right) \quad (2)$$

Generate a random sample

```
require(MASS)
```

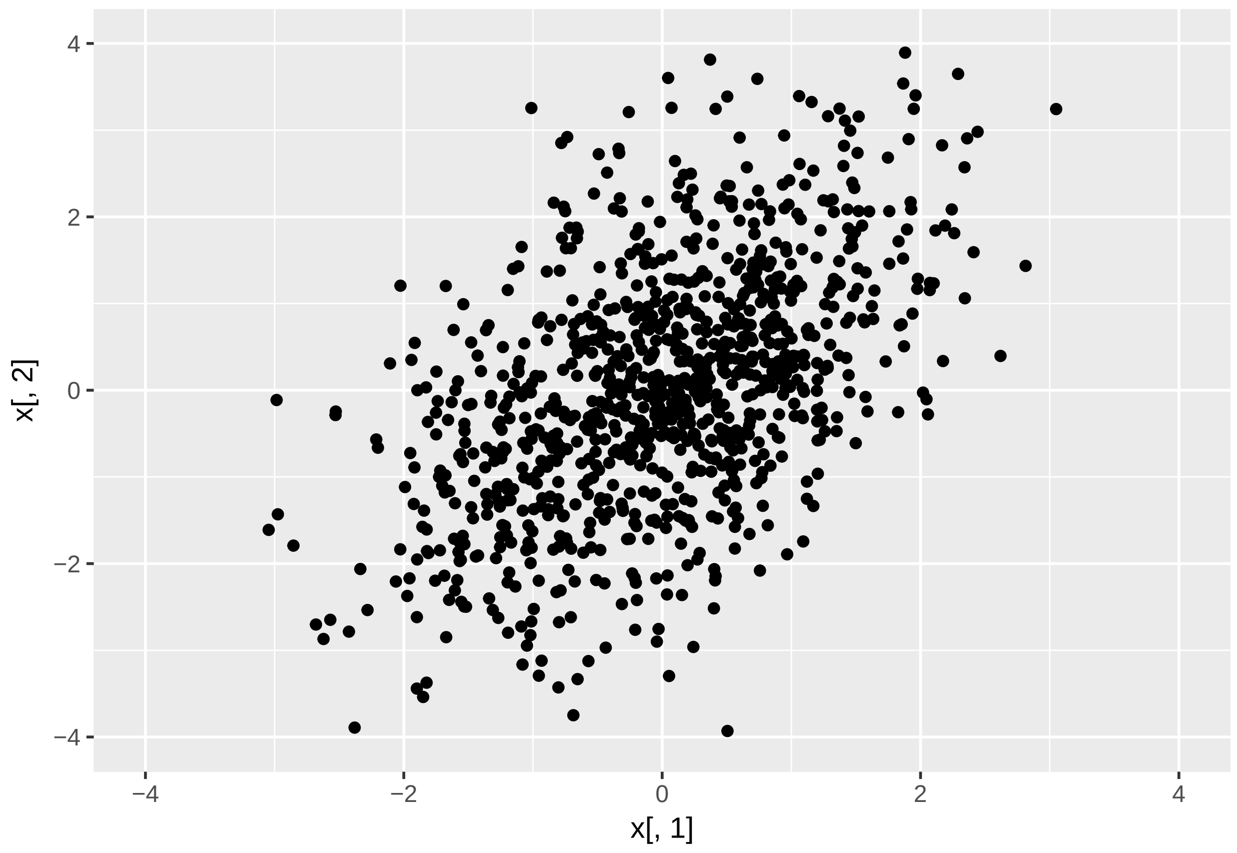
```
## Loading required package: MASS
```

```
require(ggplot2)
```

```
## Loading required package: ggplot2
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```
S = matrix(c(1,0.75,0.75,2),ncol=2)
x = mvrnorm(n = 1000, mu=c(0,0), Sigma=S)

ggplot(data.frame()) + geom_point(aes(x=x[,1],y=x[,2])) + scale_x_continuous(limits=c(-4,4))  + scale_y
```

```
## Warning: Removed 4 rows containing missing values (geom_point).
```



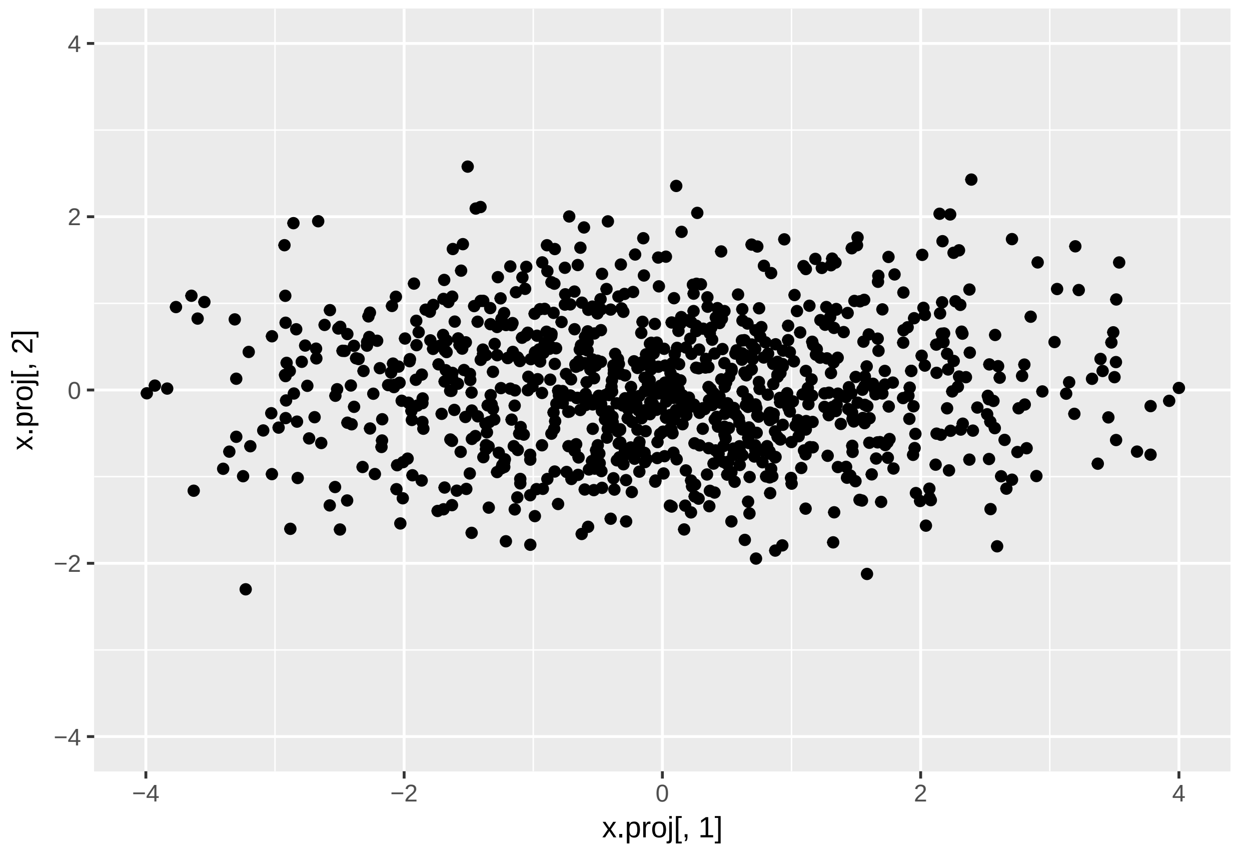
Perform PCA

```
S.sample = cov(x)

e.decomp = eigen(S.sample)

e.decomp$vectors %*% diag(e.decomp$values) %*% t(e.decomp$vectors)
```

```
##             [,1]      [,2]
## [1,] 1.0141466 0.7407353
## [2,] 0.7407353 2.0021503
```

```
x.proj = x %*% e.decomp$vectors
ggplot(data.frame()) + geom_point(aes(x=x.proj[,1],y=x.proj[,2])) + scale_x_continuous(limits=c(-4,4))
```

```
## Warning: Removed 8 rows containing missing values (geom_point).
```



```
e.decomp$values
```

```
## [1] 2.3985004 0.6177964
```

```
var(x.proj[,1])
```

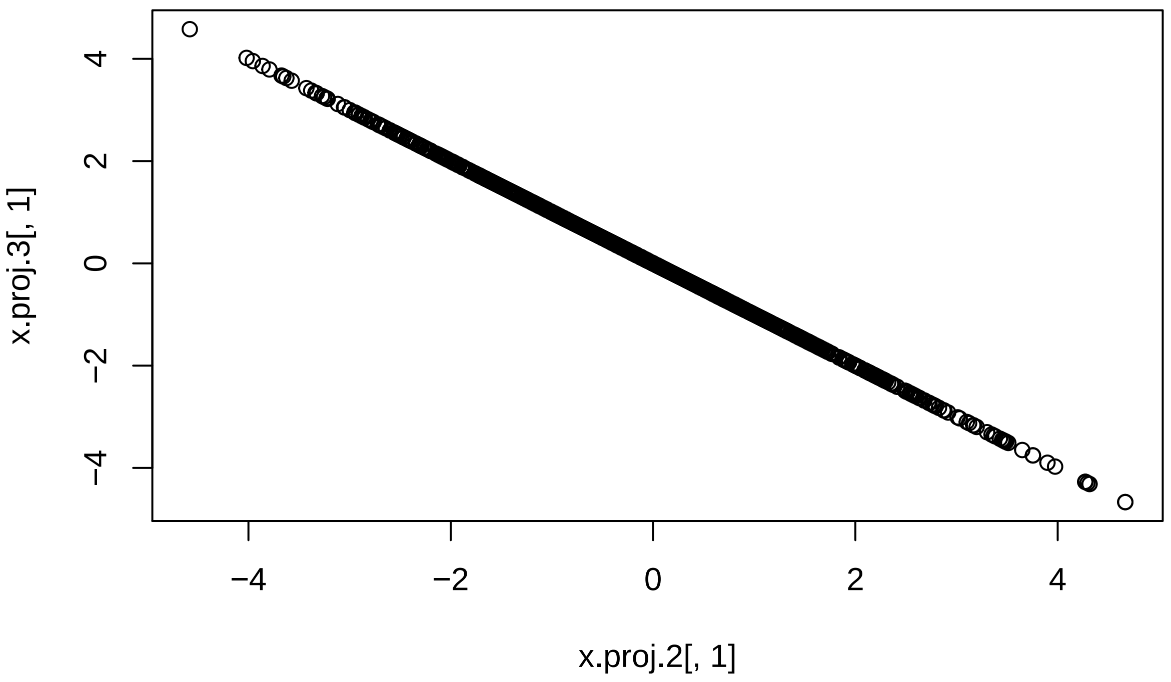
```
## [1] 2.3985
```

```
var(x.proj[,2])
```

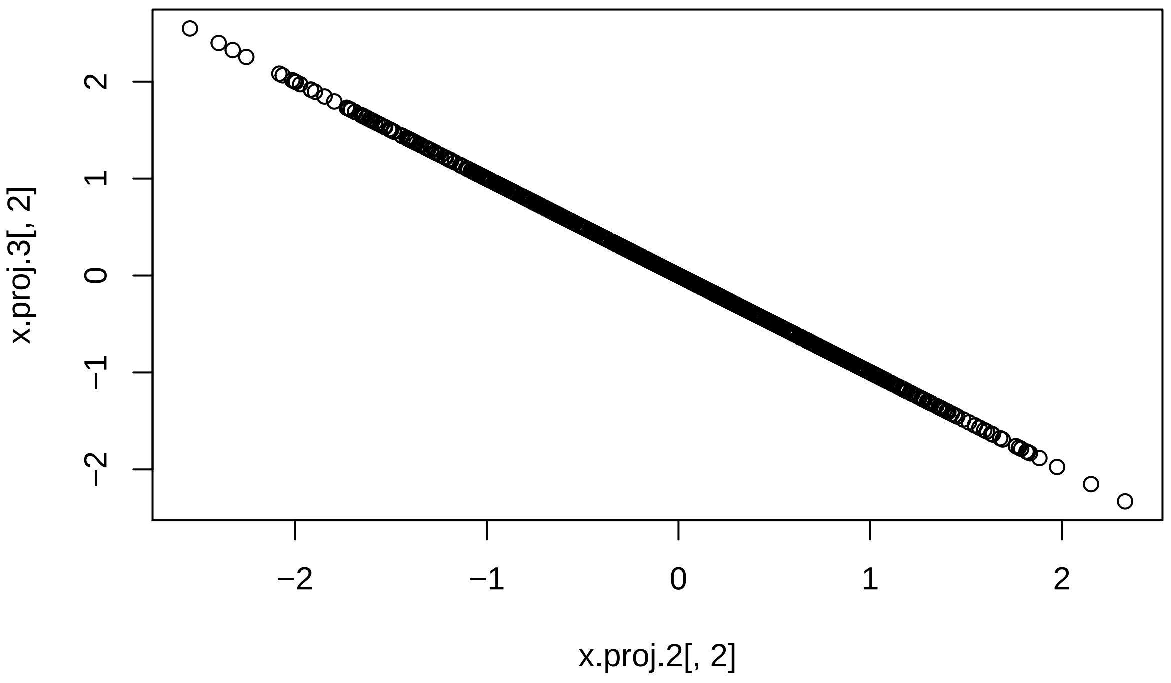
```
## [1] 0.6177964
```

Notice that the variance of the projected data matches the eigenvalues of the covariance matrix. The projection of x onto its principal components simply rotates the data.

```
svd.sample.cov = svd(S.sample)


x.center = scale(x,scale = FALSE)

svd.x = svd(x.center)


x.proj.2 = x.center %*% svd.x$v
x.proj.3 = x.center %*% svd.sample.cov$v

plot(x.proj.2[,1],x.proj.3[,1])
```



```r
plot(x.proj.2[,2],x.proj.3[,2])
```



```r
svd.x$d^2 / (1000 - 1)
```

```
## [1] 2.3985004 0.6177964
```

```r
svd.sample.cov$d
```

```
## [1] 2.3985004 0.6177964
```

The same eigenvectors and eigenvalues are computed from the matrix x.center and the covariance matrix. The singlar values σ_i of x and the eigenvalues λ_i of cov(x) are related as follows:

$$\lambda_i = \frac{\sigma_i^2}{n-1} \tag{3}$$

$$\mathrm{COV}\left[X\right] = \frac{1}{n-1} X^T X \tag{4}$$

$$X = U \Sigma V^T \tag{5}$$

$$X^T X = (U \Sigma V^T)^T (U \Sigma V^T) \tag{6}$$

$$X^T X = V \Sigma U^T U \Sigma V^T \tag{7}$$

$$X^T X = V \Sigma \Sigma V^T = V \Sigma^2 V^T \tag{8}$$

$$\frac{1}{n-1} X^T X = \frac{1}{n-1} V \Sigma^2 V^T \rightarrow \frac{1}{n-1} \Sigma^2 = \Lambda \tag{9}$$

where Λ is the diagonal matrix of eigenvalues of $\mathrm{COV}\left[X\right]$.

This is the same formula as above for the relationship between the singular values of X and the eigenvalues of its covariance matrix.

What happens if we scale X before running PCA?

```
x.scaled = scale(x)

cov.scaled.x = cov(x.scaled)
cov.scaled.x
```

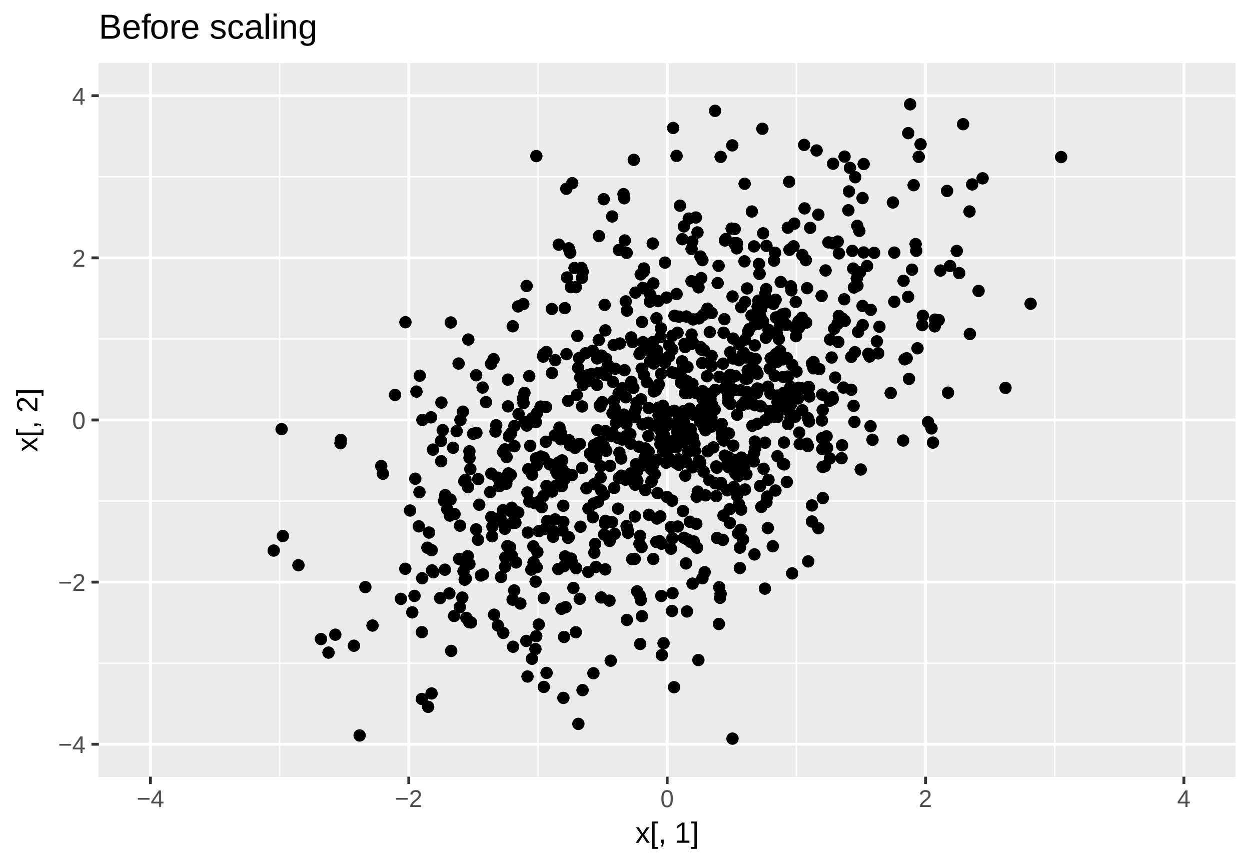
```
##           [,1]      [,2]
## [1,] 1.0000000 0.5198336
## [2,] 0.5198336 1.0000000
```

```
S.sample
```

```
##           [,1]      [,2]
## [1,] 1.0141466 0.7407353
## [2,] 0.7407353 2.0021503
```

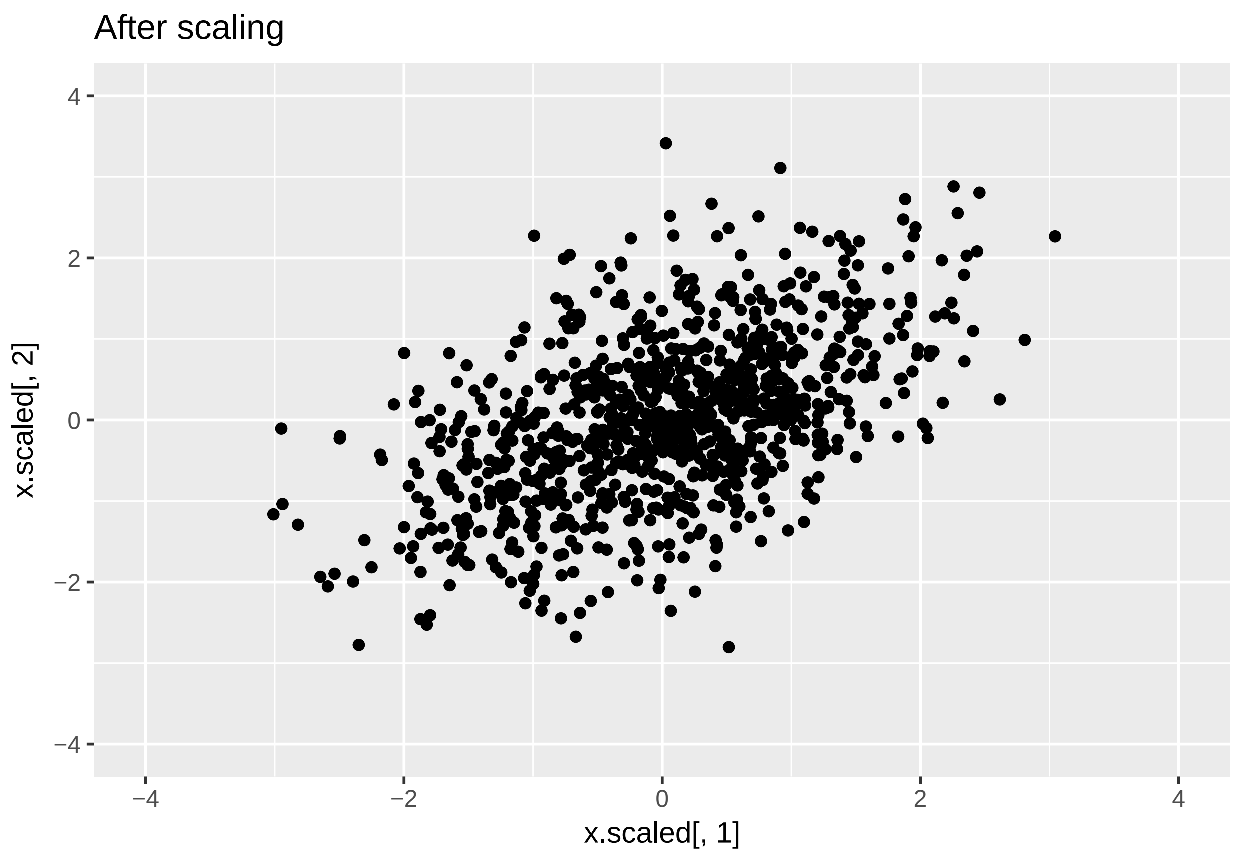
```
ggplot(data.frame()) + geom_point(aes(x=x[,1],y=x[,2])) + scale_x_continuous(limits=c(-4,4))  + scale_y
```

```
## Warning: Removed 4 rows containing missing values (geom_point).
```

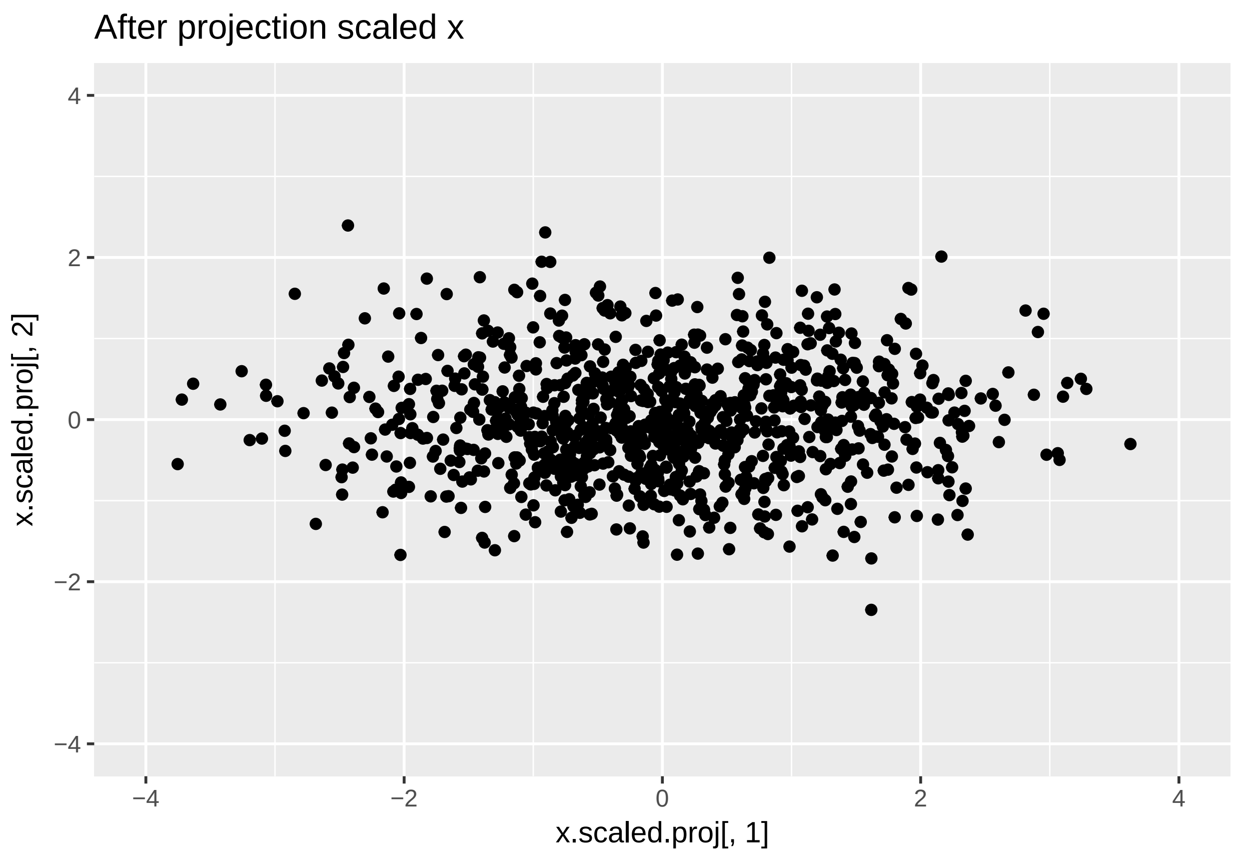


```
ggplot(data.frame()) + geom_point(aes(x=x.scaled[,1],y=x.scaled[,2])) + scale_x_continuous(limits=c(-4,4
```

After scaling



```
scaled.svd = svd(cov.scaled.x)

x.scaled.proj = x.scaled %*% scaled.svd$v
ggplot(data.frame()) + geom_point(aes(x=x.scaled.proj[,1],y=x.scaled.proj[,2])) + scale_x_continuous(li
```



```
svd.sample.cov$d
```

```
## [1] 2.3985004 0.6177964
```

```
scaled.svd$d
```

```
## [1] 1.5198336 0.4801664
```

Data reconstruction

```
x.projection = svd.sample.cov$v %*% t(x.center)
x.reconstruction = t(svd.sample.cov$v %*% x.projection)
x.reconstruction.partial = t(svd.sample.cov$v %*% rbind(x.projection[1,], 0))

cor(x.reconstruction[,1],x.center[,1])
```

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

```
cor(x.reconstruction[,2],x.center[,2])
```

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

```
cor(x.reconstruction[,1],x.reconstruction.partial[,1])
```

```
## [1] 0.725543
```

```
cor(x.reconstruction[,2],x.reconstruction.partial[,2])
```

```
## [1] 0.9650488
```

Notince that the reconstruction works perfectly, and the partial reconstruction works fairly well.

Whitening transform

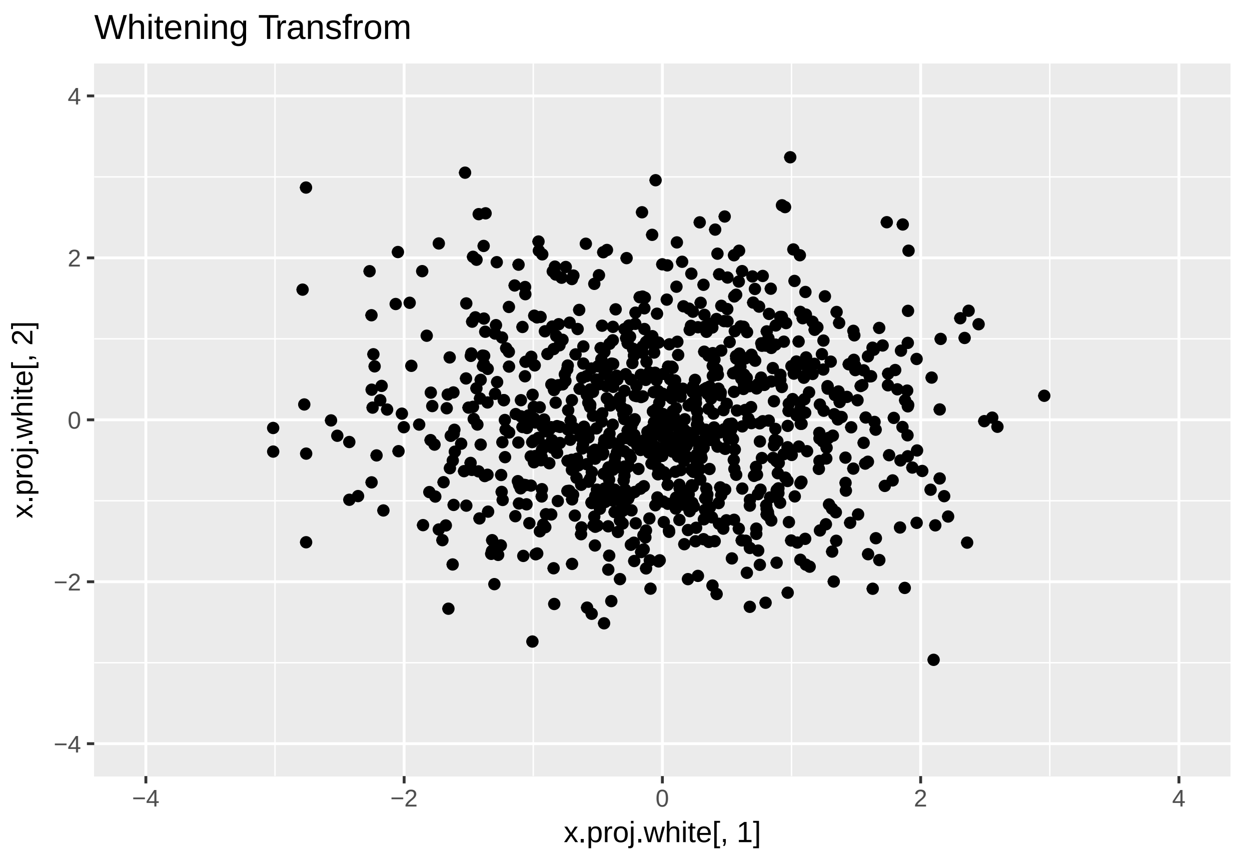
```
x.proj.white = x.center %*% svd.sample.cov$v %*% solve(diag(sqrt(svd.sample.cov$d)))
var(x.proj.white[,1])
```

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

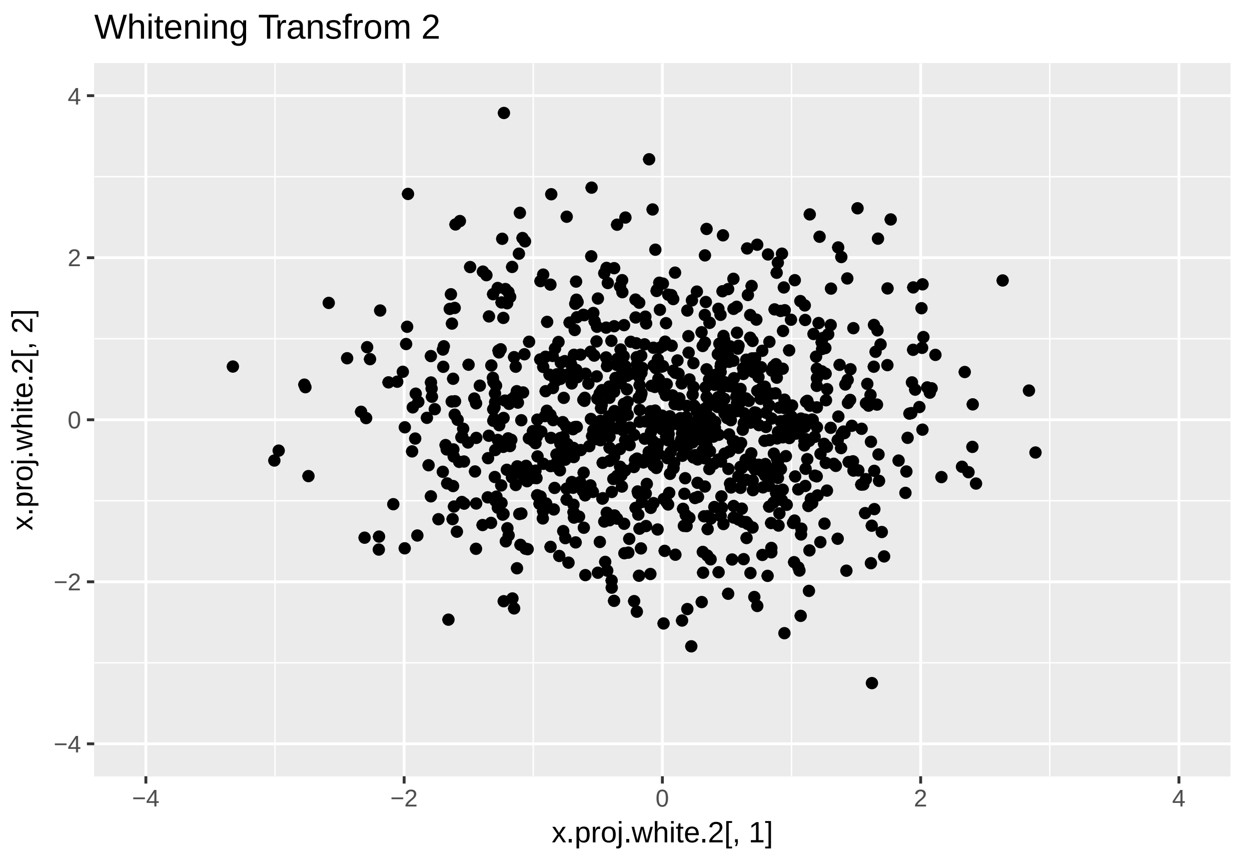
```
var(x.proj.white[,2])
```

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

```
ggplot(data.frame()) + geom_point(aes(x=x.proj.white[,1],y=x.proj.white[,2])) + scale_x_continuous(limi
```



```
w_sqrt = svd.sample.cov$v %*% diag(sqrt(svd.sample.cov$d)) %*% t(svd.sample.cov$v)
w_inv_sqrt = solve(w_sqrt)
w_inv_sqrt.2 = t(svd.sample.cov$v) %*% solve(diag(sqrt(svd.sample.cov$d))) %*% svd.sample.cov$v
x.proj.white.2= x.center %*% w_inv_sqrt.2
ggplot(data.frame()) + geom_point(aes(x=x.proj.white.2[,1],y=x.proj.white.2[,2])) + scale_x_continuous(l
```



```
cov(x.proj.white.2)
```

```
##                [,1]          [,2]
## [1,]  1.000000e+00 -3.038469e-16
## [2,] -3.038469e-16  1.000000e+00
```

```
cov(x.proj.white)
```

```
##                [,1]          [,2]
## [1,]  1.000000e+00 -8.979573e-17
## [2,] -8.979573e-17  1.000000e+00
```

The above are two different whitening transforms. Please note that:

$$\text{COV}[X] = V\Lambda V^T \tag{10}$$

In a diagonal matrix Λ we can take the square roots by just taking the square root of the diagonal elements

$$\text{COV}[X] = (V\Lambda^{1/2}V^T)^2 = V\Lambda^{1/2}(V^TV)\Lambda^{1/2}V^T = V\Lambda V^T \tag{11}$$

Since, $V^TV = I$. That is $V^T = V^{-1}$.

$$\text{COV}[X]^{1/2} = V\Lambda^{1/2}V^T \tag{12}$$

Now $(ABC)^-1 = C^{-1}B^{-1}A^{-1}$. Therefore

$$\text{COV}[X]^{-1/2} = (V\Lambda^{1/2}V^T)^{-1} = V^{-1}\Lambda^{-1/2}(V^{-1})^-1 = V^T\Lambda^{-1/2}V \tag{13}$$

Since $V^T = V^{-1}$.

Mahalanobis distance^2

The Mahalanobis of a centered vector from the origin is:

$$D_M(x)^2 = x^T S^{-1} x \tag{14}$$

where S is the covariance matrix of the vector x.

```r
t(x.center[1,]) %*% solve(S.sample) %*% x.center[1,]
```

```
##            [,1]
## [1,] 0.3627102
```

```r
sum(x.proj.white[1,]^2)
```

```
## [1] 0.3627102
```

```r
sum(x.proj.white.2[1,]^2)
```

```
## [1] 0.3627102
```

```r
x.1.proj.scaled = x.center[1,] %*% svd.sample.cov$v %*% solve(diag(sqrt(svd.sample.cov$d)))
sum(x.1.proj.scaled^2)
```

```
## [1] 0.3627102
```

Mahalanobis distance^2 is the equal to the $\|x\|^2$ in the Whitened space.

The norm of a vector

$$\|x\|^2 = \sum x_i^2 = x^T x \tag{15}$$

$$(x^T)^T = x \tag{16}$$

$$D_M(x)^2 = x^T \mathrm{COV}[X]^{-1} x \tag{17}$$

$$\mathrm{COV}[X]^{-1} = (V \Lambda V^T)^{-1} = V^T \Lambda^{-1} V \tag{18}$$

The Mahalanobis distance^2 can be written as

$$D_M(x)^2 = x^T V^T \Lambda^{-1} V x = x^T V^T \Lambda^{-1/2} \Lambda^{-1/2} V x = (x^T V^T \Lambda^{-1/2})(\Lambda^{-1/2} V x) = (\Lambda^{-1/2} V x)^T (\Lambda^{-1/2} V x) \tag{19}$$

$$= (\Lambda^{-1/2} V x)^T (\Lambda^{-1/2} V x) = \|\Lambda^{-1/2} V x\|^2 = \|x^T V^T \Lambda^{-1/2}\|^2 = D_M(x)^2 \tag{20}$$

The final equality $\|x^T V^T \Lambda^{-1/2}\|^2$ shows that the $D_M(x)^2$ is the same as projecting the x onto its principal components, then scaling each axis by the square root of its eigenvalue (if the eigenvalue is the variance then the sqrt(eigenvalue) is like the standard deviation) and finally taking the norm of the scaled projected x. A eigenvalue is the variance of the data projected onto its corresponding eigenvector, so to scale it you divide by the standard deviation.

What about PCA and SVD in the case where there are more variables than observations?

Generate random samples. Assume we have $n = 100$ subjects and $m = 200$ snps.

```
n = 100
m = 50000
z = rbinom(n,size = 2, prob = 0.5)
X = matrix(rep(0,n * m),ncol=n)
snp_afs_0 = runif(m,0.01,0.5)
snp_afs_1 = runif(m,0.25,0.75)
snp_afs_2 = c(snp_afs_0[1:(m/2)],snp_afs_1[(m/2 + 1):m])
for(i in 1:n) {
  x_i = NULL
  if(z[i] == 0) {
    x_i = rbinom(m,size=2,prob=snp_afs_0)
  } else if(z[i] == 1) {
    x_i = rbinom(m,size=2,prob=snp_afs_1)
  } else {
    x_i = rbinom(m,size=2,prob=snp_afs_2)
  }
  X[,i] = x_i
}
```

Standardize by subtracting off the row means and dividing by the standard deviation. Also compute covariance matrix.

```
X_std = t(scale(t(X)))
ind_cov = cov(X_std)
ind_cor_x = cor(X)

#approx covariance matrix
Xt_X = t(X_std) %*% X_std * 1/(n-1)

dim(ind_cov)
```

```
## [1] 100 100
```

```
dim(Xt_X)
```

```
## [1] 100 100
```

```
sign(Xt_X[1:10,1:10]) == sign(ind_cov[1:10,1:10])
```

```
##        [,1]  [,2]  [,3] [,4] [,5]  [,6]  [,7] [,8]  [,9] [,10]
##  [1,]  TRUE FALSE  TRUE TRUE TRUE FALSE  TRUE TRUE FALSE  TRUE
##  [2,] FALSE  TRUE FALSE TRUE TRUE  TRUE FALSE TRUE  TRUE  TRUE
##  [3,]  TRUE FALSE  TRUE TRUE TRUE FALSE  TRUE TRUE FALSE  TRUE
##  [4,]  TRUE  TRUE  TRUE TRUE TRUE  TRUE  TRUE TRUE  TRUE  TRUE
##  [5,]  TRUE  TRUE  TRUE TRUE TRUE  TRUE  TRUE TRUE  TRUE  TRUE
##  [6,] FALSE  TRUE FALSE TRUE TRUE  TRUE FALSE TRUE  TRUE  TRUE
##  [7,]  TRUE FALSE  TRUE TRUE TRUE FALSE  TRUE TRUE FALSE  TRUE
##  [8,]  TRUE  TRUE  TRUE TRUE TRUE  TRUE  TRUE TRUE  TRUE  TRUE
##  [9,] FALSE  TRUE FALSE TRUE TRUE  TRUE FALSE TRUE  TRUE  TRUE
## [10,]  TRUE  TRUE  TRUE TRUE TRUE  TRUE  TRUE TRUE  TRUE  TRUE
```

```
Xt_X[1:10,1:10] / ind_cov[1:10,1:10]
```

```
##               [,1]       [,2]       [,3]      [,4]      [,5]       [,6]
##  [1,]   511.1433 -2299.8931   552.5964  741.8059  718.7674 -3109.7082
##  [2,] -2299.8931   625.4373 -2473.9409 1348.9850 1342.6063  1309.7657
##  [3,]   552.5964 -2473.9409   511.1046  735.6752  728.6359 -2101.6812
##  [4,]   741.8059  1348.9850   735.6752  533.5258 1334.3593  1408.8509
##  [5,]   718.7674  1342.6063   728.6359 1334.3593  531.9334  1354.4483
##  [6,] -3109.7082  1309.7657 -2101.6812 1408.8509 1354.4483   629.4193
##  [7,]   554.9157 -2286.1969   554.8381  738.0696  711.8100 -3069.2554
##  [8,]   763.6156  1373.6485   725.3805 1555.9220 1399.0801  1269.7613
##  [9,] -4070.5740  1262.3947 -1880.5719 1341.1359 1408.6822  1270.2287
## [10,]   762.1407  1279.3938   806.7408 1726.5715 1479.4782  1355.4573
##               [,7]      [,8]       [,9]     [,10]
##  [1,]   554.9157  763.6156 -4070.5740  762.1407
##  [2,] -2286.1969 1373.6485  1262.3947 1279.3938
##  [3,]   554.8381  725.3805 -1880.5719  806.7408
##  [4,]   738.0696 1555.9220  1341.1359 1726.5715
##  [5,]   711.8100 1399.0801  1408.6822 1479.4782
##  [6,] -3069.2554 1269.7613  1270.2287 1355.4573
##  [7,]   511.5332  727.1693 -4948.3299  750.4688
##  [8,]   727.1693  533.4899  1316.5059 1880.2777
##  [9,] -4948.3299 1316.5059   623.4179 1210.7870
## [10,]   750.4688 1880.2777  1210.7870  533.6553
```

Now let us compare the eigenvalues of each

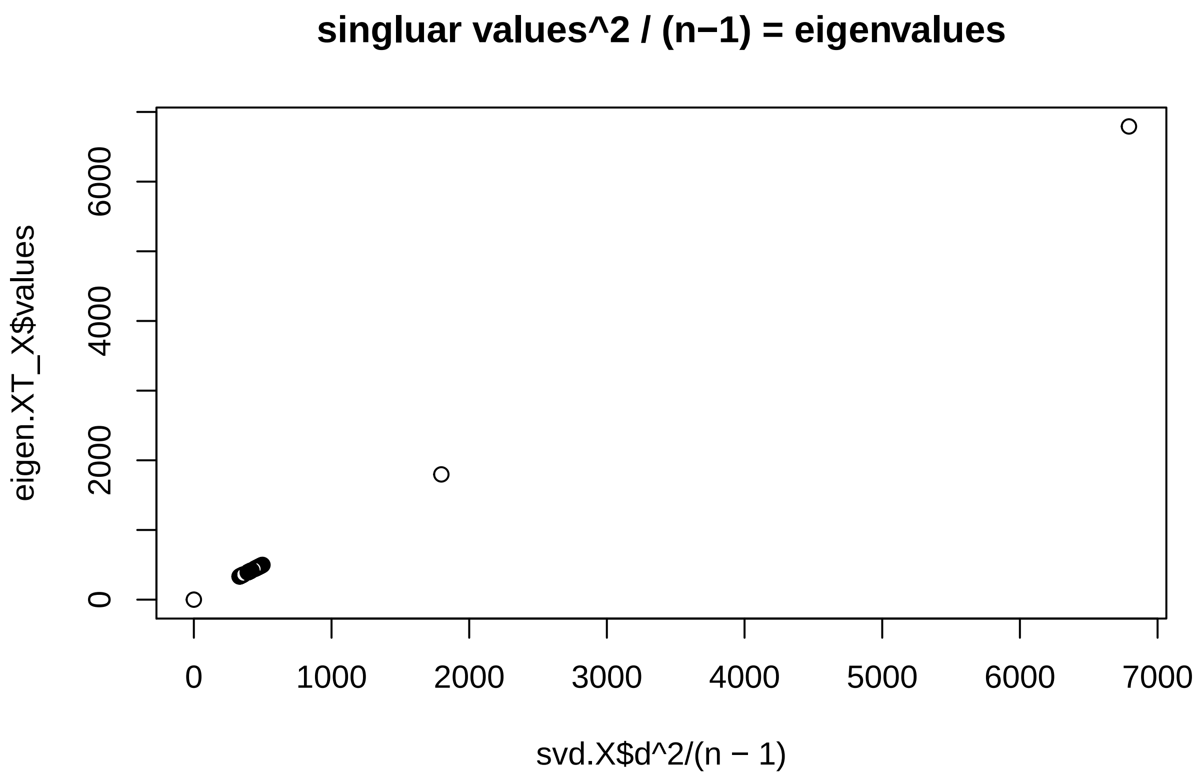
```
eigen.XT_X = eigen(Xt_X)
eigen.ind_cov = eigen(ind_cov)
eigen.ind_cor = eigen(cor(X_std))
eigen.ind_cor_x = eigen(ind_cor_x)
svd.X = svd(X_std)

color = function(x) {
  a_func = function(a) {
    if(a == 0) {
      return("black")
    } else if(a == 1) {
      return("red")
    } else {
      return("blue")
    }
  }
  sapply(x,FUN=a_func)
}

(svd.X$d^2 / (n - 1))[1:10]
```

```
##  [1] 6792.0129 1797.7053  498.3740  495.2265  493.4469  491.1551  490.8398
##  [8]  490.4760  488.6024  487.0333
```

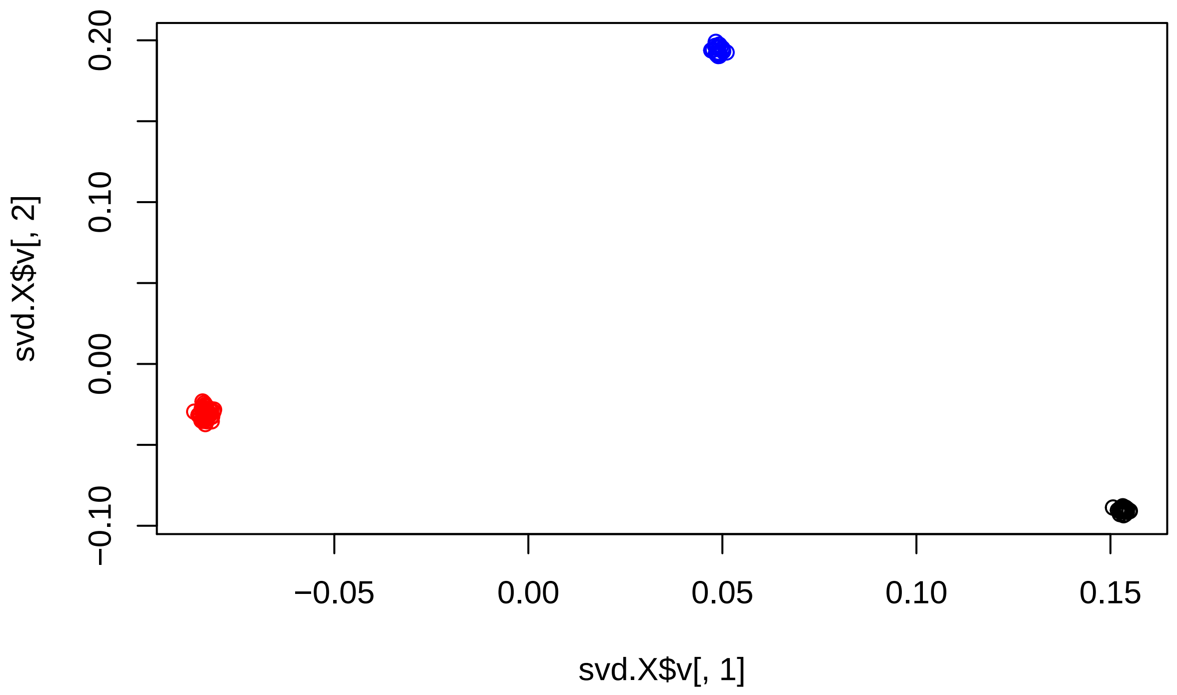
```
eigen.XT_X$values[1:10]
```

```
##  [1] 6792.0129 1797.7053  498.3740  495.2265  493.4469  491.1551  490.8398
##  [8]  490.4760  488.6024  487.0333
```

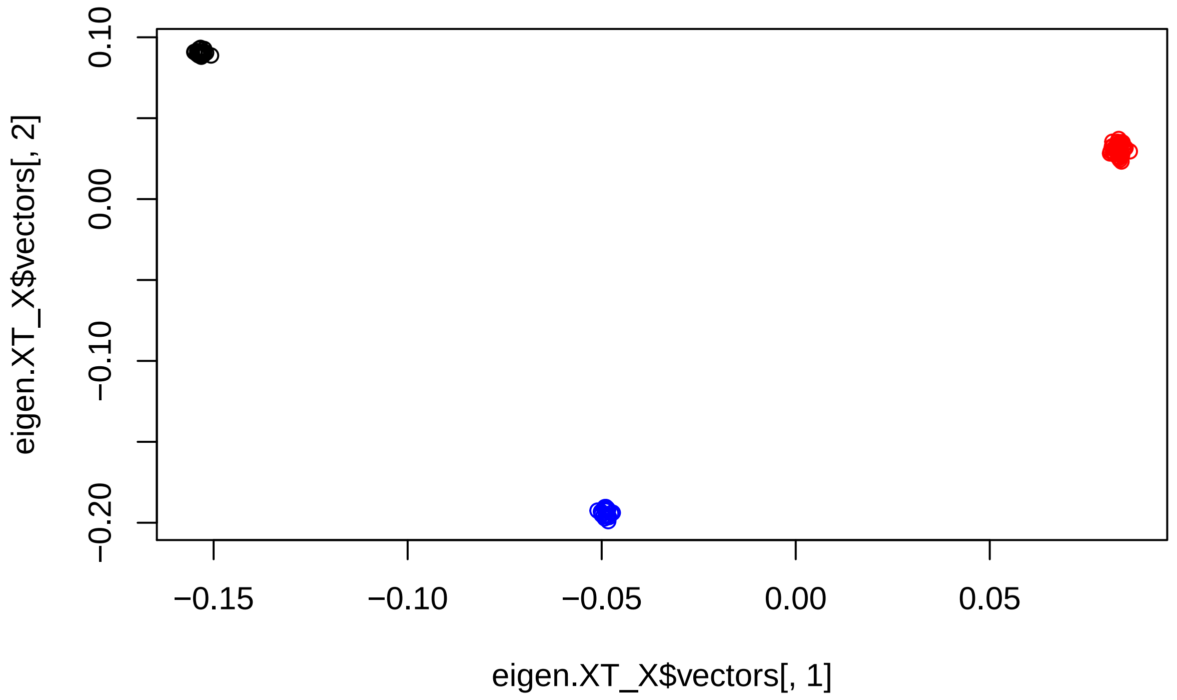
```r
plot(svd.X$d^2 / (n - 1),eigen.XT_X$values,main="singluar values^2 / (n-1) = eigenvalues")
```



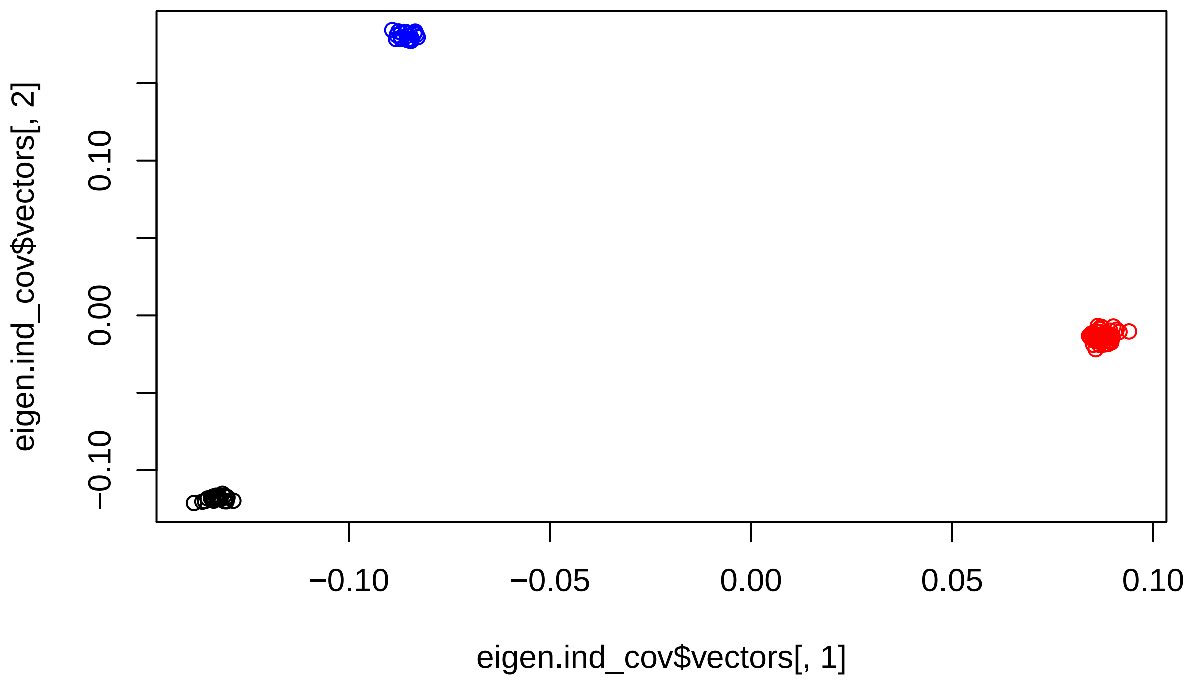
```r
plot(svd.X$v[,1],svd.X$v[,2],col=color(z))
```



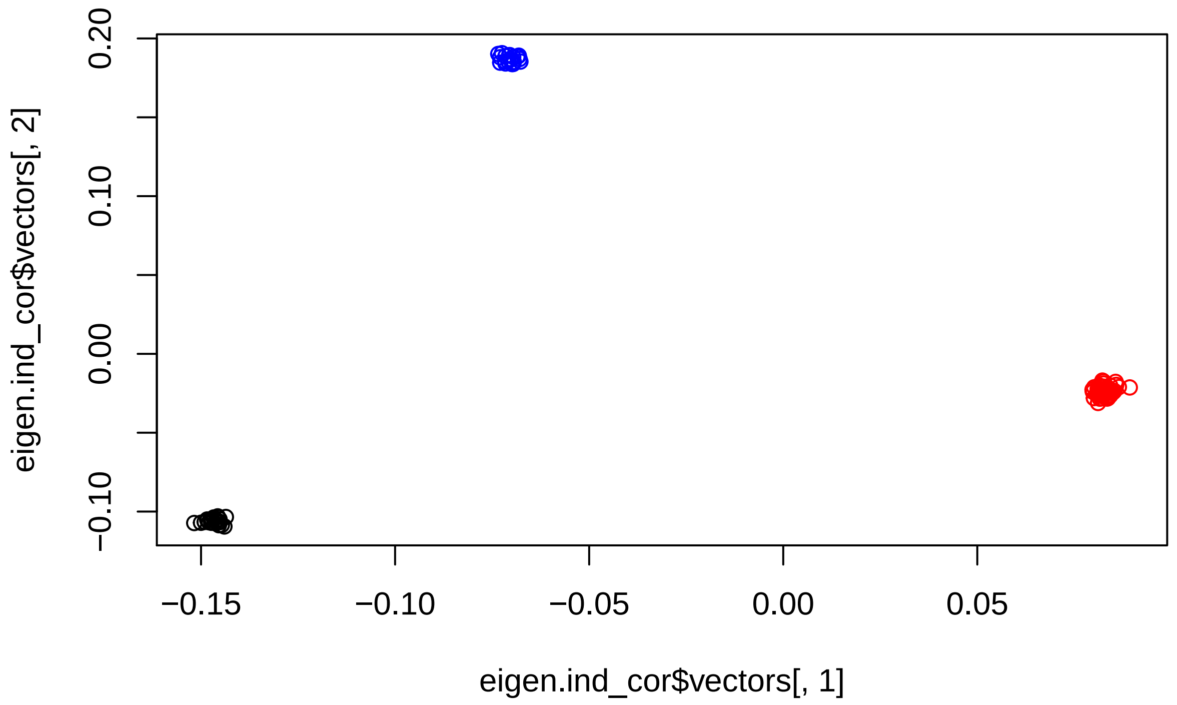
```r
plot(eigen.XT_X$vectors[,1],eigen.XT_X$vectors[,2],col=color(z))
```



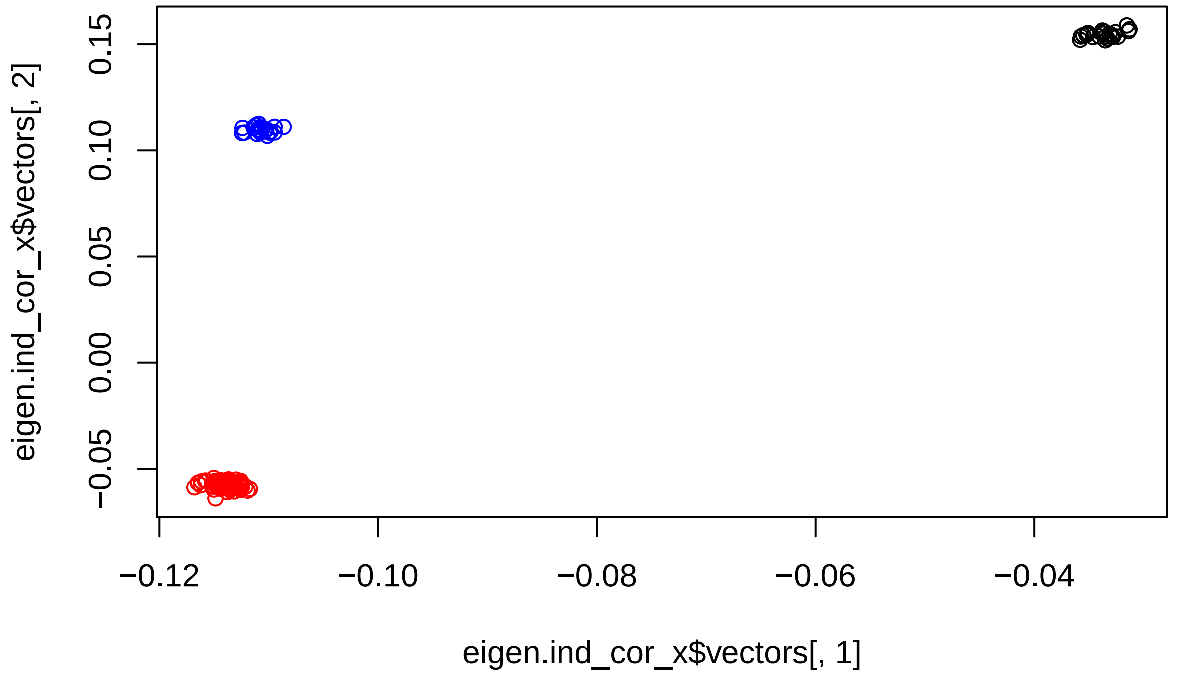
```
plot(eigen.ind_cov$vectors[,1],eigen.ind_cov$vectors[,2],col=color(z))
```



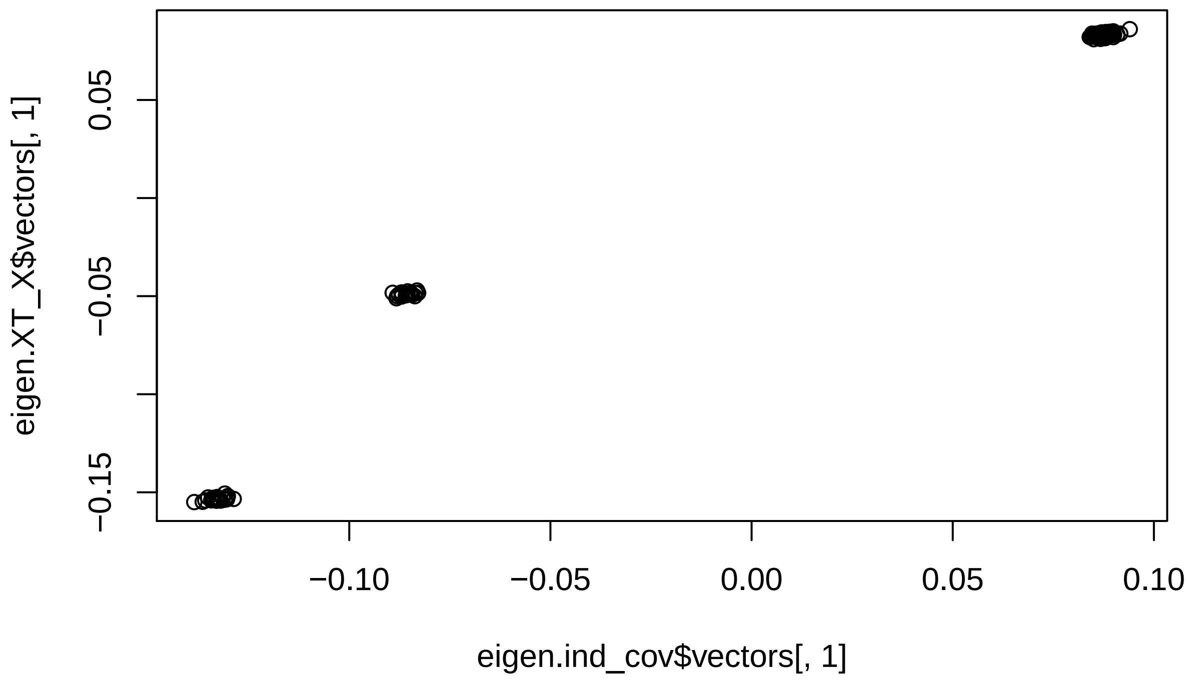
```
plot(eigen.ind_cor$vectors[,1],eigen.ind_cor$vectors[,2],col=color(z))
```



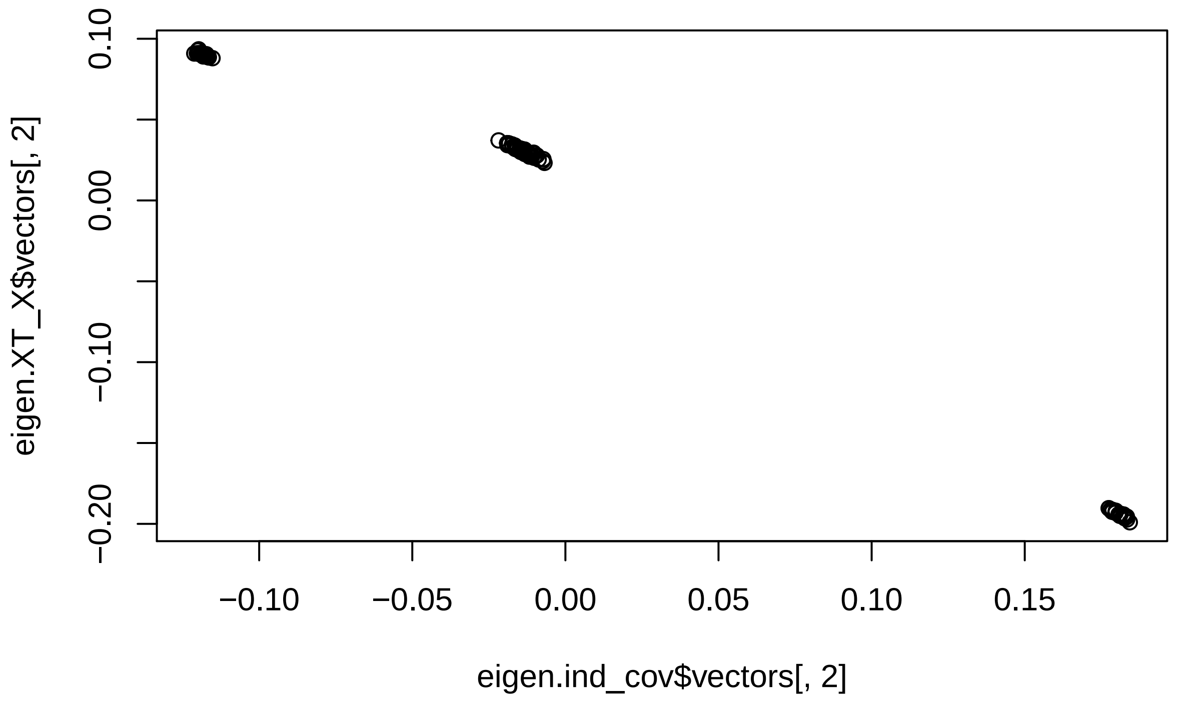
```
plot(eigen.ind_cor_x$vectors[,1],eigen.ind_cor_x$vectors[,2],col=color(z))
```



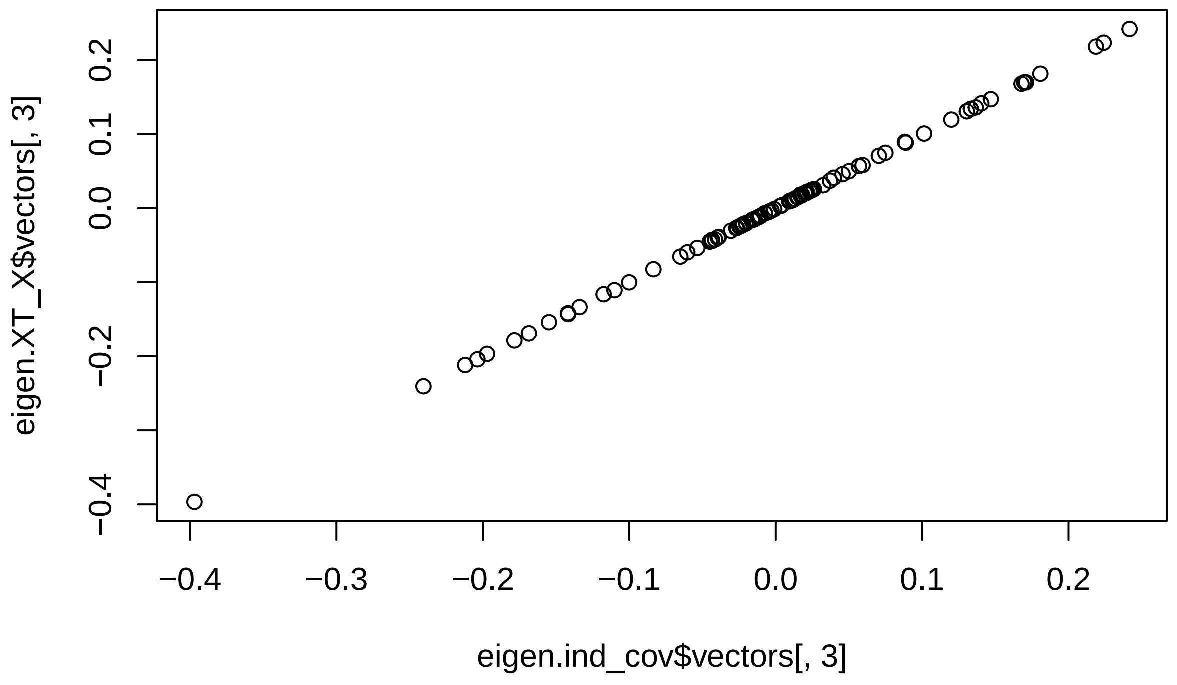
```
plot(eigen.ind_cov$vectors[,1],eigen.XT_X$vectors[,1])
```



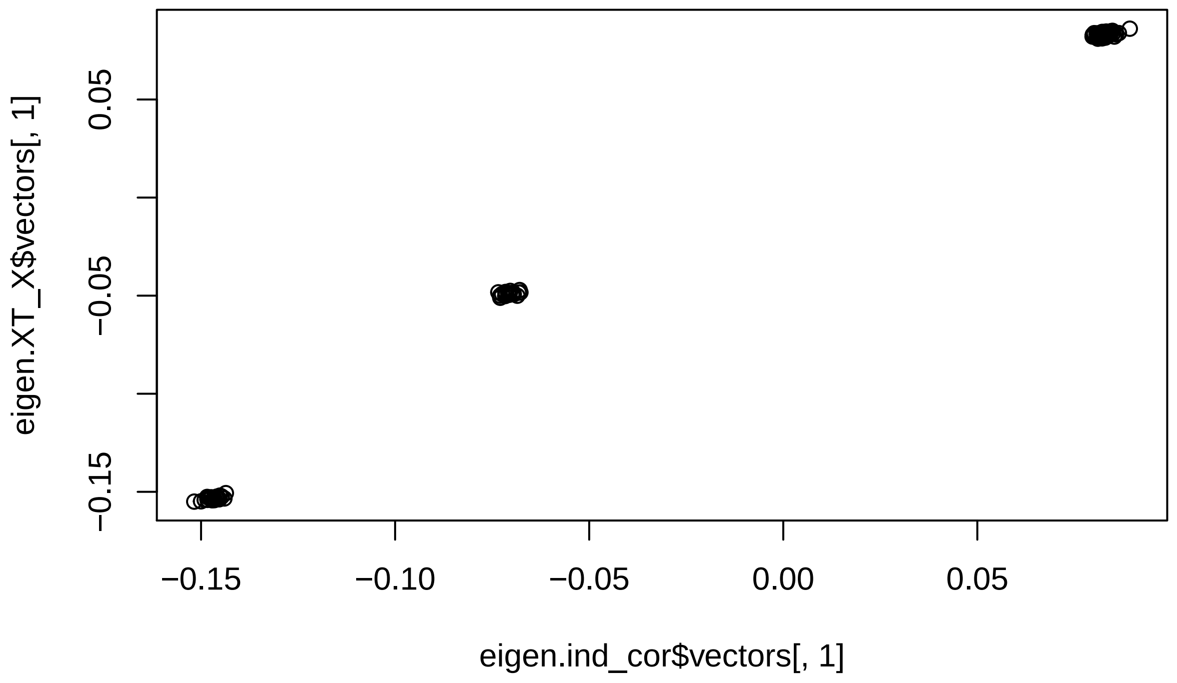
```r
plot(eigen.ind_cov$vectors[,2],eigen.XT_X$vectors[,2])
```



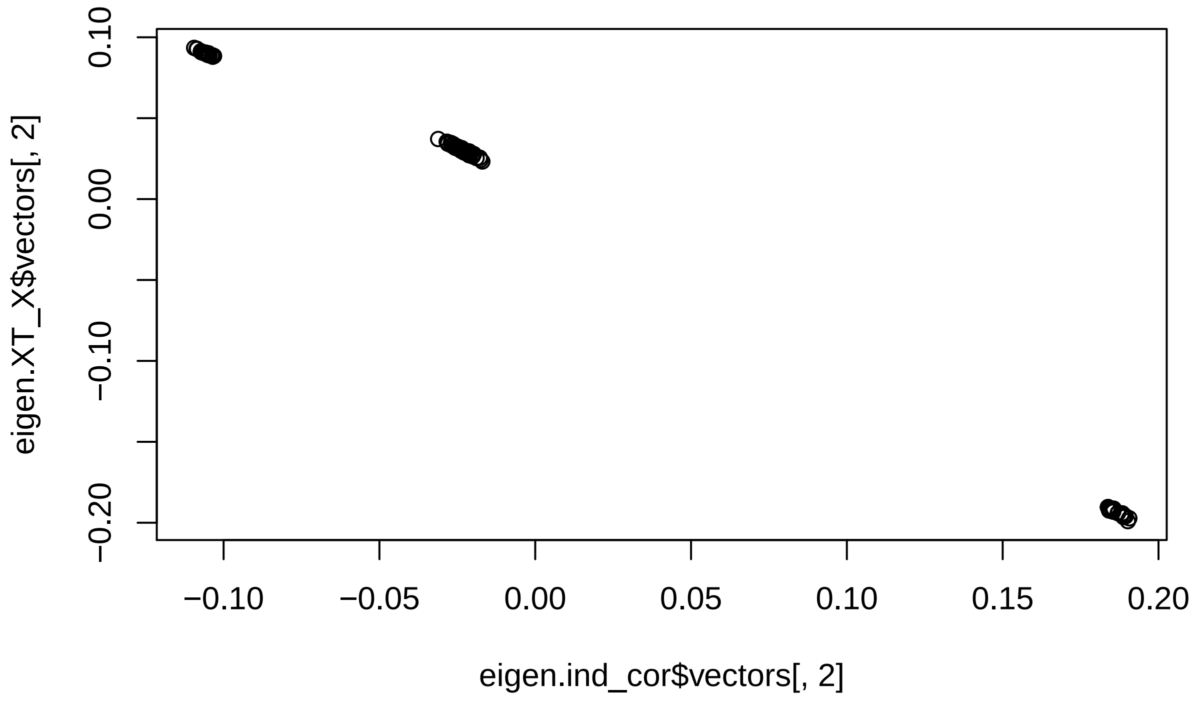
```r
plot(eigen.ind_cov$vectors[,3],eigen.XT_X$vectors[,3])
```



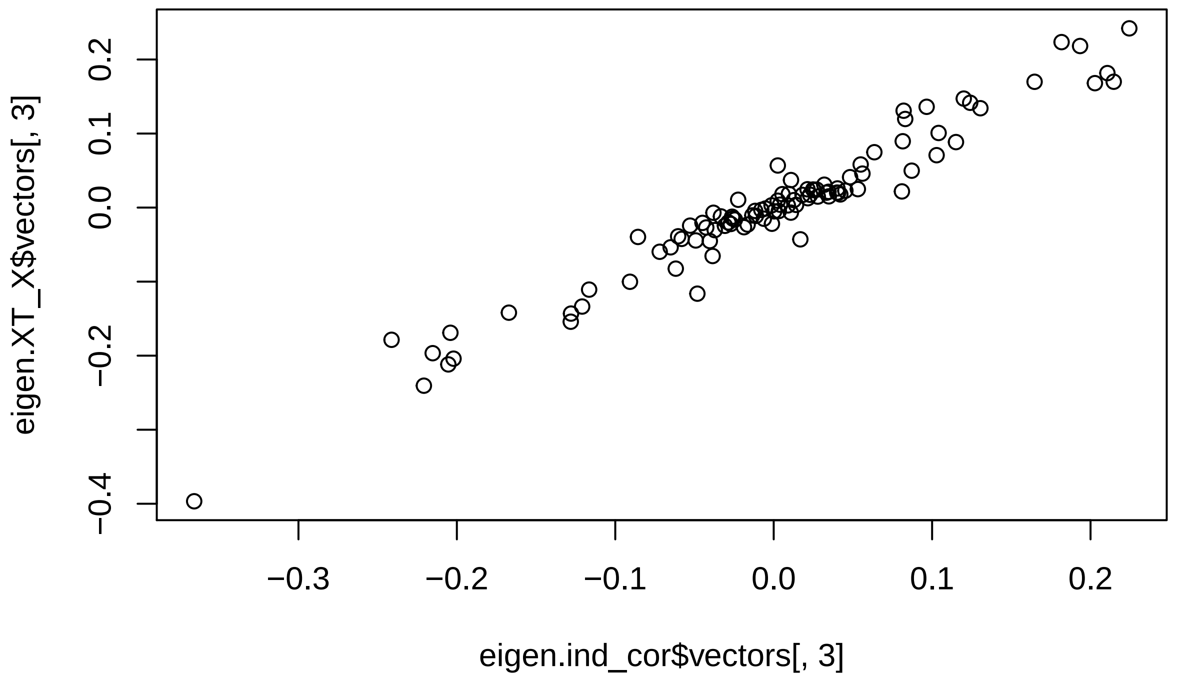
```
plot(eigen.ind_cor$vectors[,1],eigen.XT_X$vectors[,1])
```



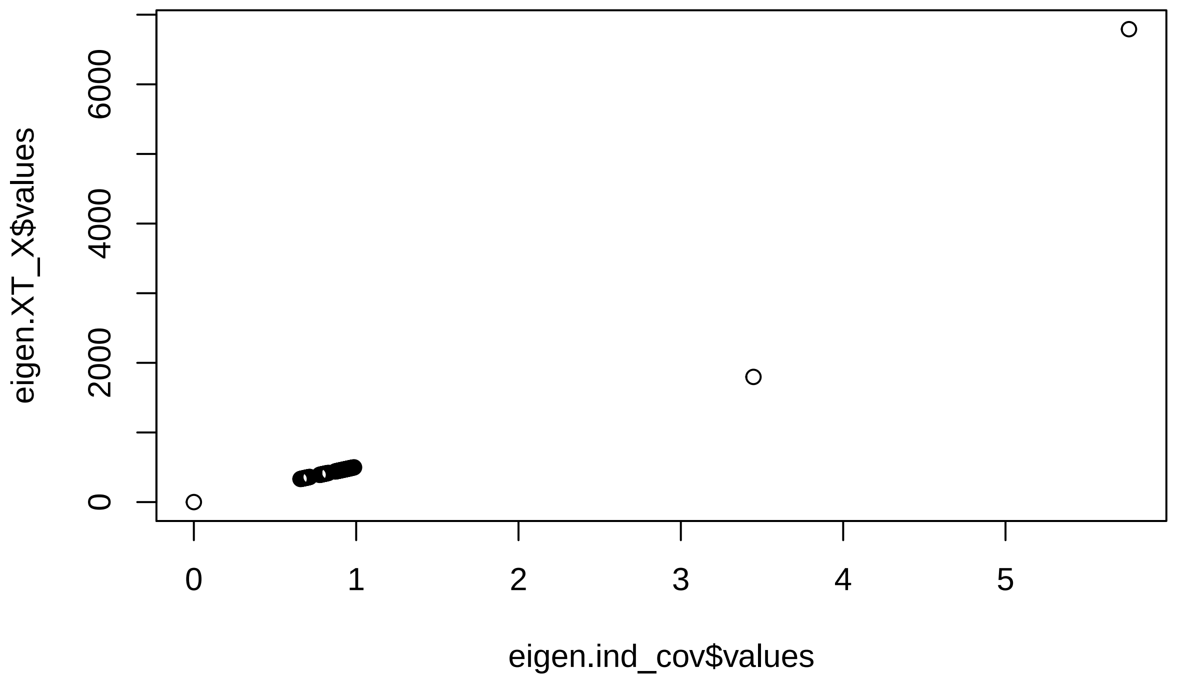
```
plot(eigen.ind_cor$vectors[,2],eigen.XT_X$vectors[,2])
```



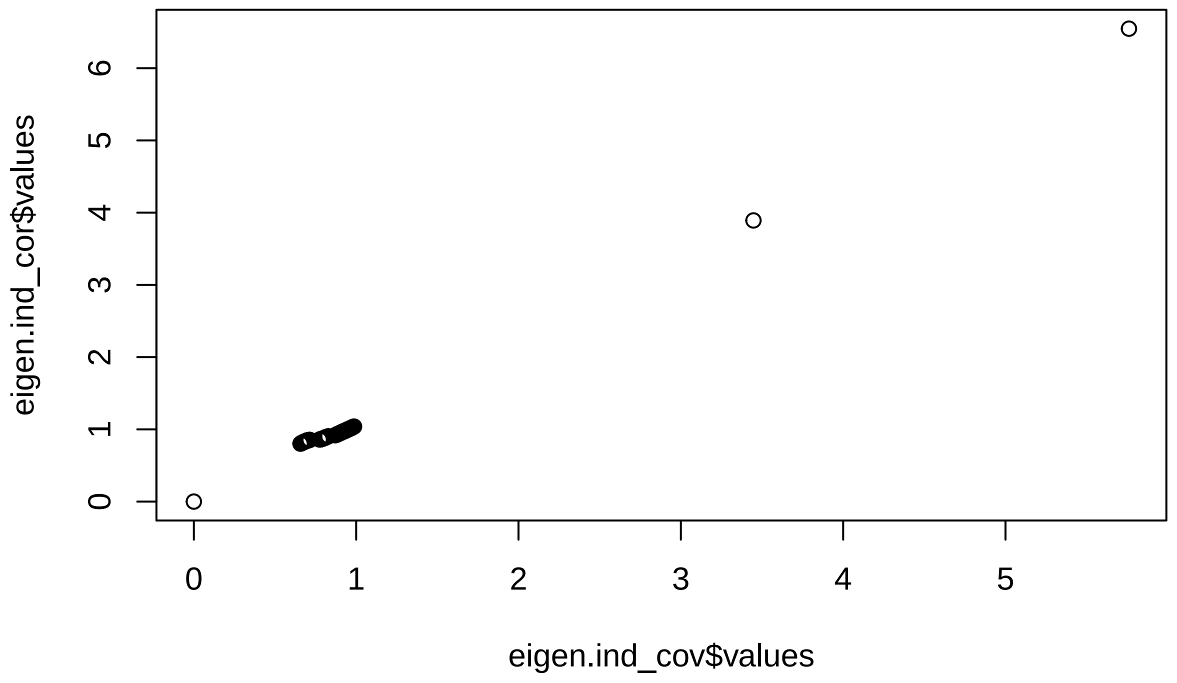
```r
plot(eigen.ind_cor$vectors[,3],eigen.XT_X$vectors[,3])
```



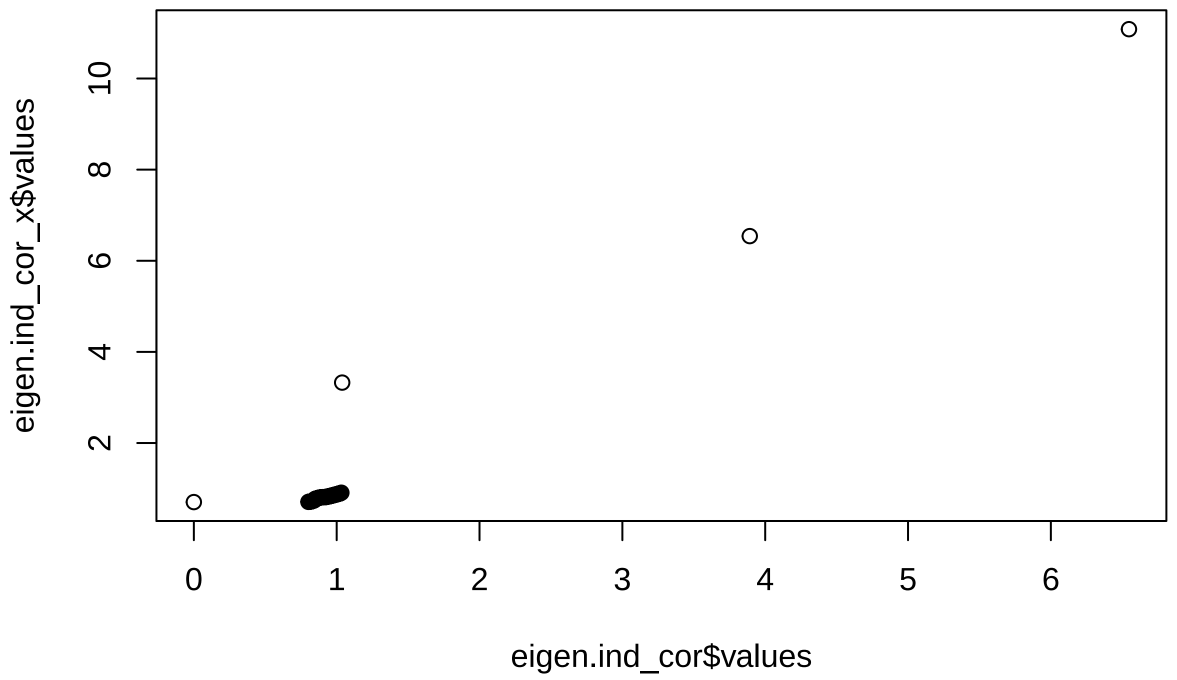
```r
plot(eigen.ind_cov$values,eigen.XT_X$values)
```



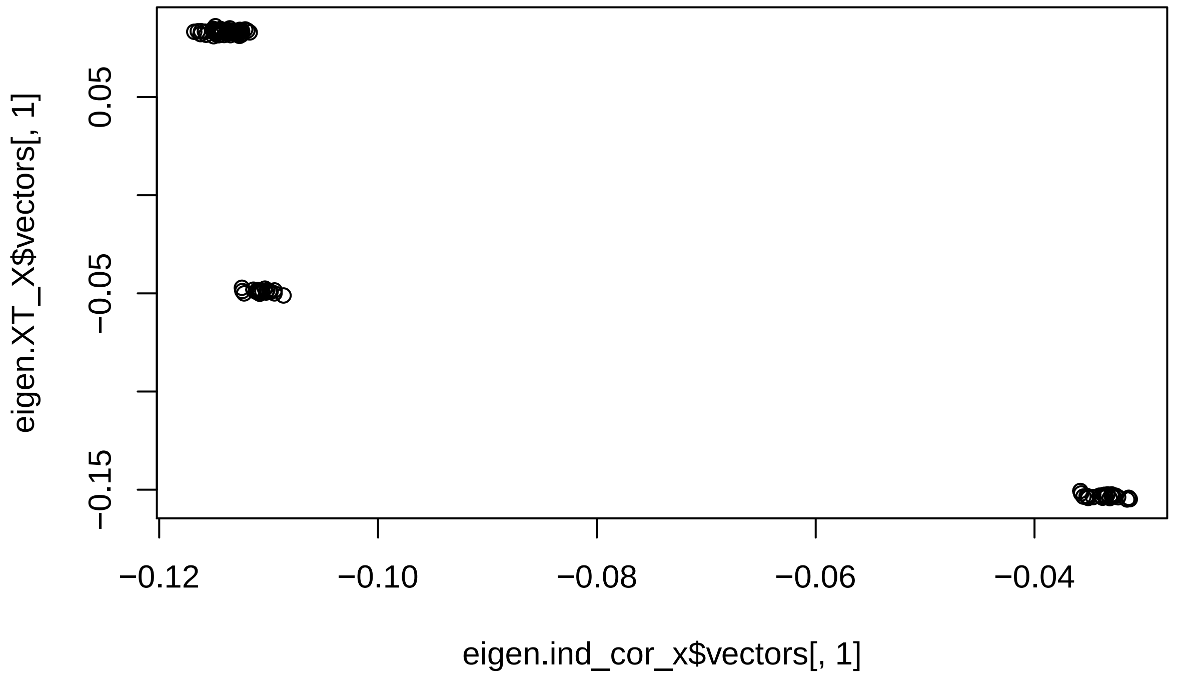
```r
plot(eigen.ind_cov$values,eigen.ind_cor$values)
```



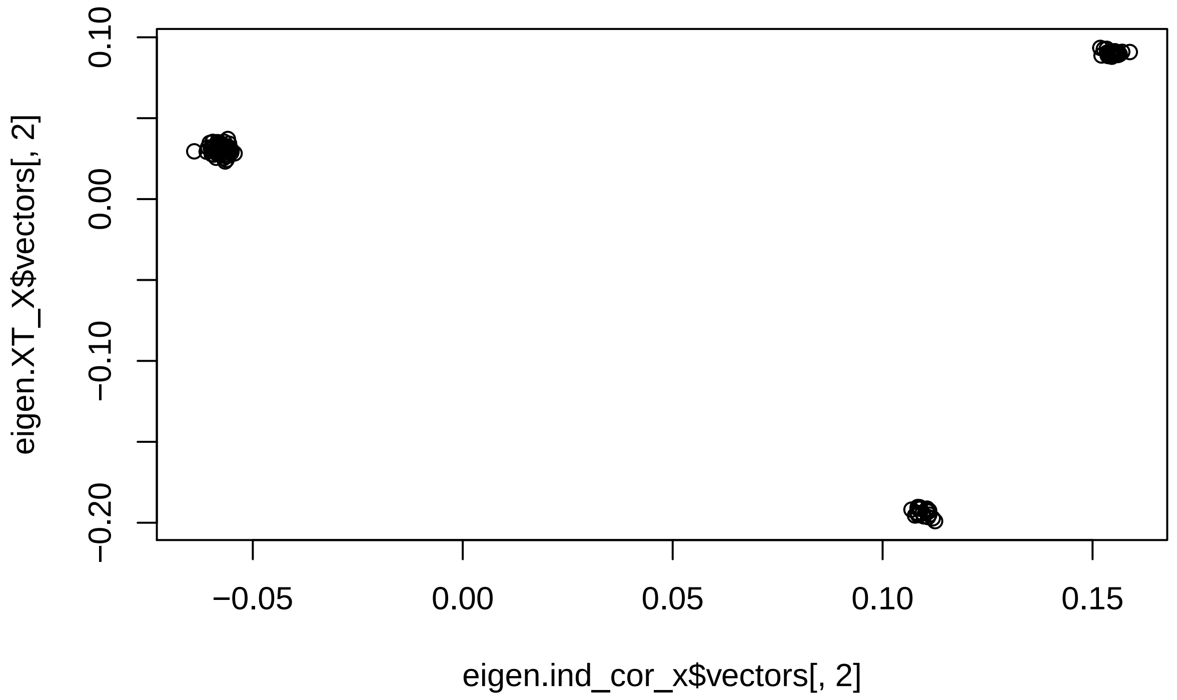
```r
plot(eigen.ind_cor$values,eigen.ind_cor_x$values)
```



```r
plot(eigen.ind_cor_x$vectors[,1],eigen.XT_X$vectors[,1])
```



```r
plot(eigen.ind_cor_x$vectors[,2],eigen.XT_X$vectors[,2])
```



The eigenvalues from svd match those from the eigenvalues from the matrix Xt_X

Now let us see how