Kun Zhou 204688165

1)

```
X.toba = as.matrix(tobacco[,4:9])
Y.toba = as.matrix(tobacco[,1:3])
beta_tls = tls(X.toba, Y.toba)$b
beta_ols = solve(crossprod(X.toba),crossprod(X.toba,Y.toba))
beta_tls
```

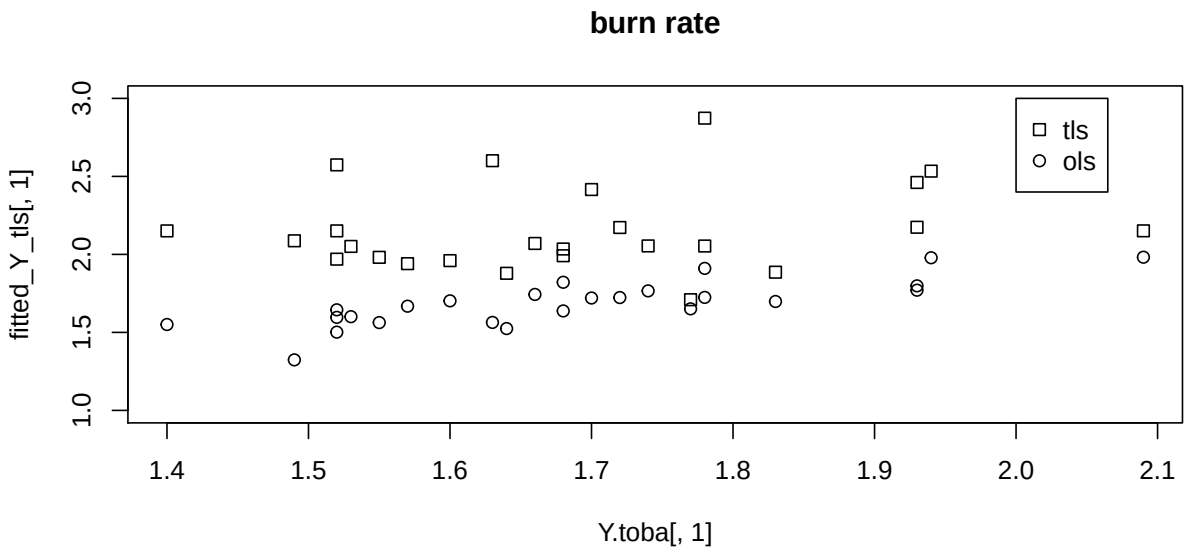
```
##            [,1]       [,2]        [,3]
## [1,] -0.6885734 -4.8393868  -1.9389344
## [2,] -0.2168204  0.9861221  -0.4926901
## [3,]  1.0326846  1.4811444   1.8528465
## [4,] -2.6255490 49.3147396 -12.5578216
## [5,]  0.4701792 -0.9367463  -0.2278278
## [6,]  1.0324576  0.5379732  10.7824045
```

```
beta_ols
```

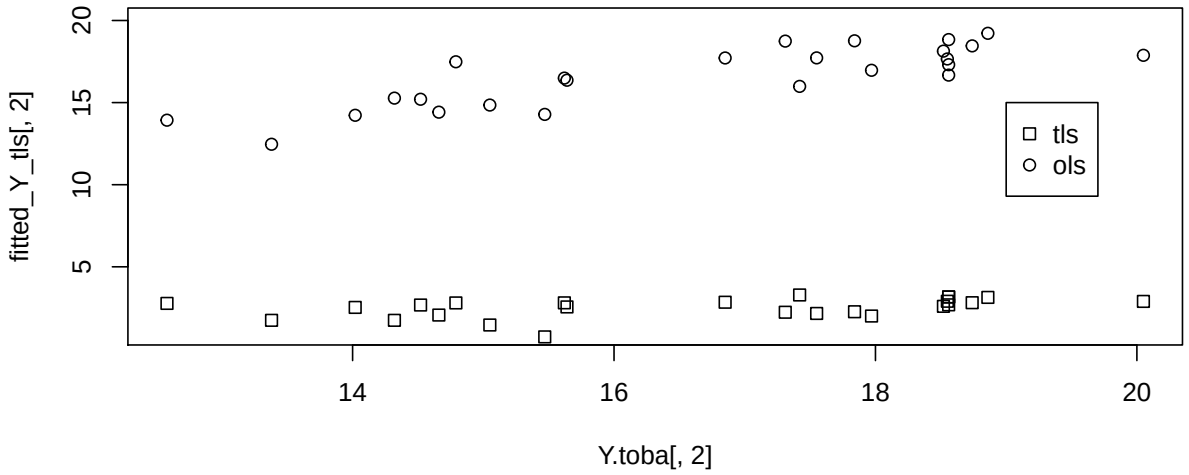
```
##                     Y1.BurnRate Y2.PercentSugar Y3.PercentNicotine
## X1.PercentNitrogen   -0.07582591       -5.649931         0.70496833
## X2.PercentChlorine   -0.16691944        1.260679        -0.27103036
## X3.PercentPotassium   0.55874631        4.165826        -0.07807883
## X4.PercentPhosphorus  0.15123535       21.770434        -1.62045353
## X5.PercentCalcium     0.30984483        1.874391         0.17137157
## X6.PercentMagnesium  -0.18082559       -1.110807         1.73038010
```

Phosphorus speeds up burn rate and magnesium slows down burn rate in tls() method, while ols() method makes a opposite conclusion. Simialry, calcium and magnesium have opposite effects for percent sugar in two methods and nitrogen, potassium and phosphorus have opposite effects for percent nicotine in two methods. For tls(), phosphorus contribute a lot to percen sugar while for ols(), the contribution is halved. Magnesium and phosphorus effect percent nicotine in tls() more than in ols().

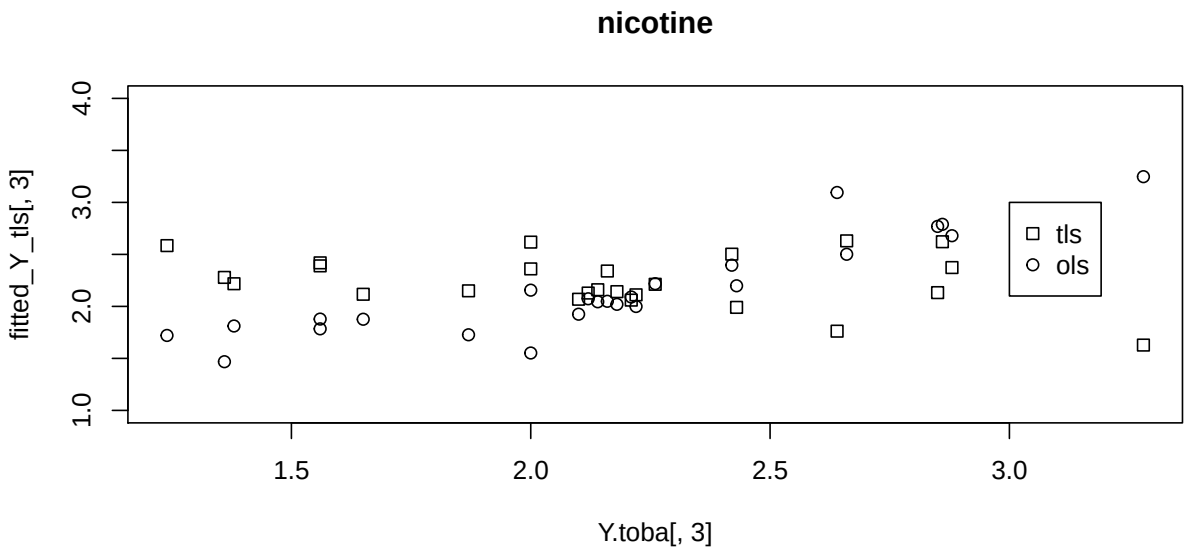
```
fitted_Y_tls = tls(X.toba, Y.toba)$z
fitted_Y_ols = X.toba %*% beta_ols
plot(Y.toba[,1], fitted_Y_tls[,1], pch=0, ylim = c(1, 3), main = "burn rate")
points(Y.toba[,1], fitted_Y_ols[,1], pch=1)
legend(x = 2.0, y = 3.0, legend = c("tls", "ols"), pch= c(0, 1))
```




```
plot(Y.toba[,2], fitted_Y_tls[,2], pch=0,  ylim = c(1, 20),main = "sugar")
points(Y.toba[,2], fitted_Y_ols[,2], pch=1)
legend(x = 19, y = 15, legend = c("tls", "ols"), pch= c(0, 1))
```




```
plot(Y.toba[,3], fitted_Y_tls[,3], pch=0,  ylim = c(1, 4),main = "nicotine")
points(Y.toba[,3], fitted_Y_ols[,3], pch=1)
legend(x = 3, y = 3, legend = c("tls", "ols"), pch= c(0, 1))
```



For the fitted value, fitted burn rates of tls() are overall higher than those of ols(), but they don't differ much. Fitted percent sugar of tls() are much smaller than thsoe of ols(). Fitted percent nicotine are almost same for both methods.

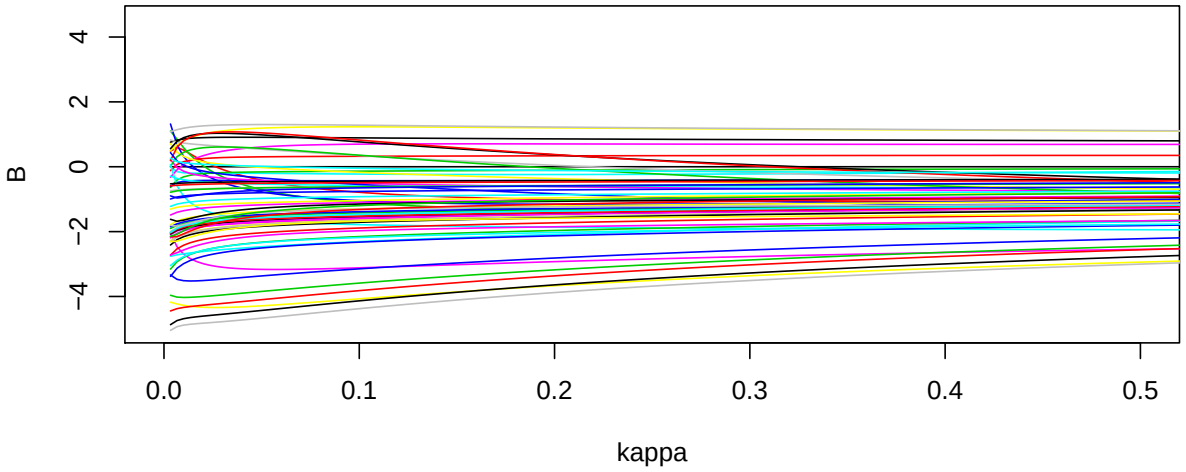
2)

$$X^* = \begin{pmatrix} X \\ \sqrt{k} I_r \end{pmatrix} \qquad Y^* = \begin{pmatrix} Y \\ 0 \end{pmatrix}$$

$$\widehat{\beta} = (X^{*t} X^*)^{-1} X^{*t} Y^* = \left[\begin{pmatrix} X^t & \sqrt{k} I_r \end{pmatrix} \begin{pmatrix} X \\ \sqrt{k} I_r \end{pmatrix} \right]^{-1} \begin{pmatrix} X^t & \sqrt{k} I_r \end{pmatrix} \begin{pmatrix} Y \\ 0 \end{pmatrix} = (X^t X + k I_r)^{-1} X^t Y$$

3)

```
X = as.matrix(pet[,1:268])
Y = as.matrix(pet[,269, drop = FALSE])
X.center = scale(X, center = T, scale = F)
Y.center = scale(Y, center = T, scale = F)
ridgeTrace(X.center, Y.center, 1, 300)
```



I center the data and include intercept in the ridge regression. But I don't penalize the intercept. When κ is increasing, the estimated parameters are becoming more stable. From the figure, when κ reaches 0.15, the first 60 parameters have been stable, so if we continue to increase κ, we only increase biased errors while decrease variance little. When κ is between 0 and 0.05, some paraemeters' traces are wave lines.

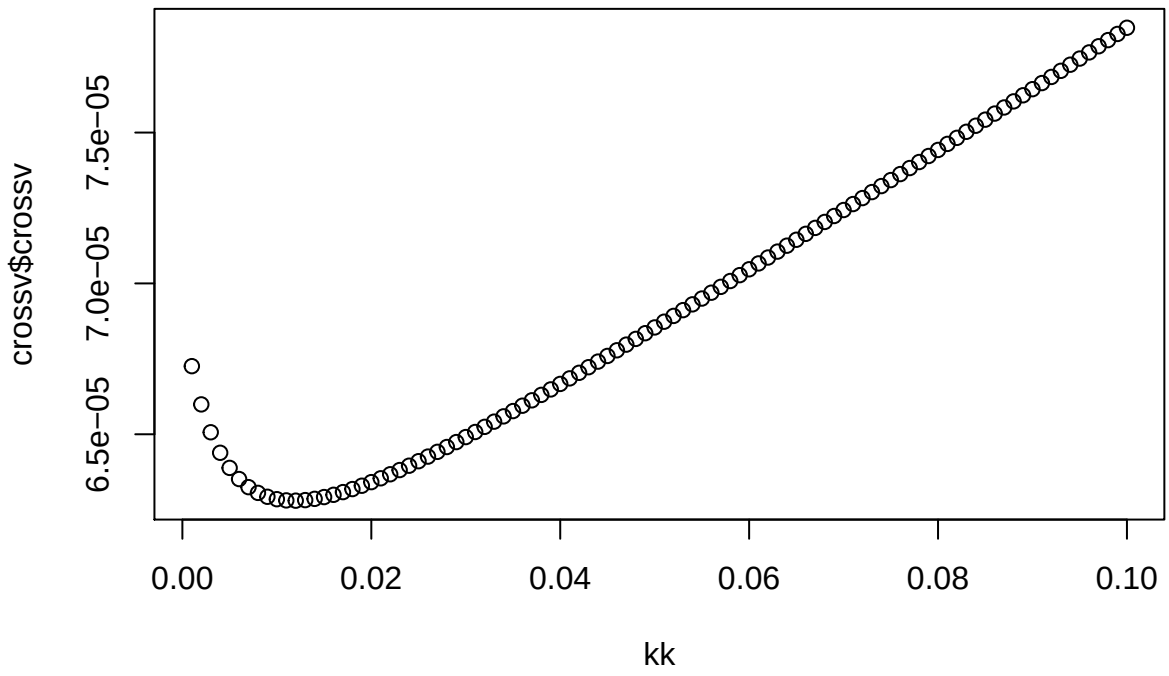
4)

```
ridgeParameter<-function(x, y, kmax, nmax) #x, y are matrices
{
  kk<-seq(0,kmax,length=nmax+1)[-1]
  crossv = rep(NaN, nmax)
  sd.cv = rep(NaN, nmax)
  for(i in 1:nmax)
  {
    sse = matrix(NaN, nrow = nrow(y), ncol = ncol(y))
    for(j in 1:nrow(x))
    {
      xTemp = x[-j, ,drop = FALSE]
      yTemp = y[-j, ,drop = FALSE]
      sse[j,] = y[j,,drop = FALSE] - cbind(1,x[j,,drop = FALSE]) %*%
        ridge(xTemp, yTemp, kk[i])$b
    }
    #cross-validation
    crossv[i] = mean(sse^2)
    #one stnadard error
    sd.cv[i] = sd(sse^2)/nrow(x)
  }
  return(list(crossv = crossv, sd.cv = sd.cv))
}
X.scale = scale(X)
Y.scale = scale(Y)
nmax = 100
kmax = 0.1
```

```
kk = seq(0,kmax,length=nmax+1)[-1]
crossv = ridgeParameter(X.scale, Y.scale, kmax, nmax)
minIndex = match(min(crossv$crossv),crossv$crossv)[1]
kk[minIndex]
```

```
## [1] 0.012
```

```
plot(kk, crossv$crossv)
```



I include the intercept and scale X and Y, so the parameters may be different. The ridege parameter is 0.012.

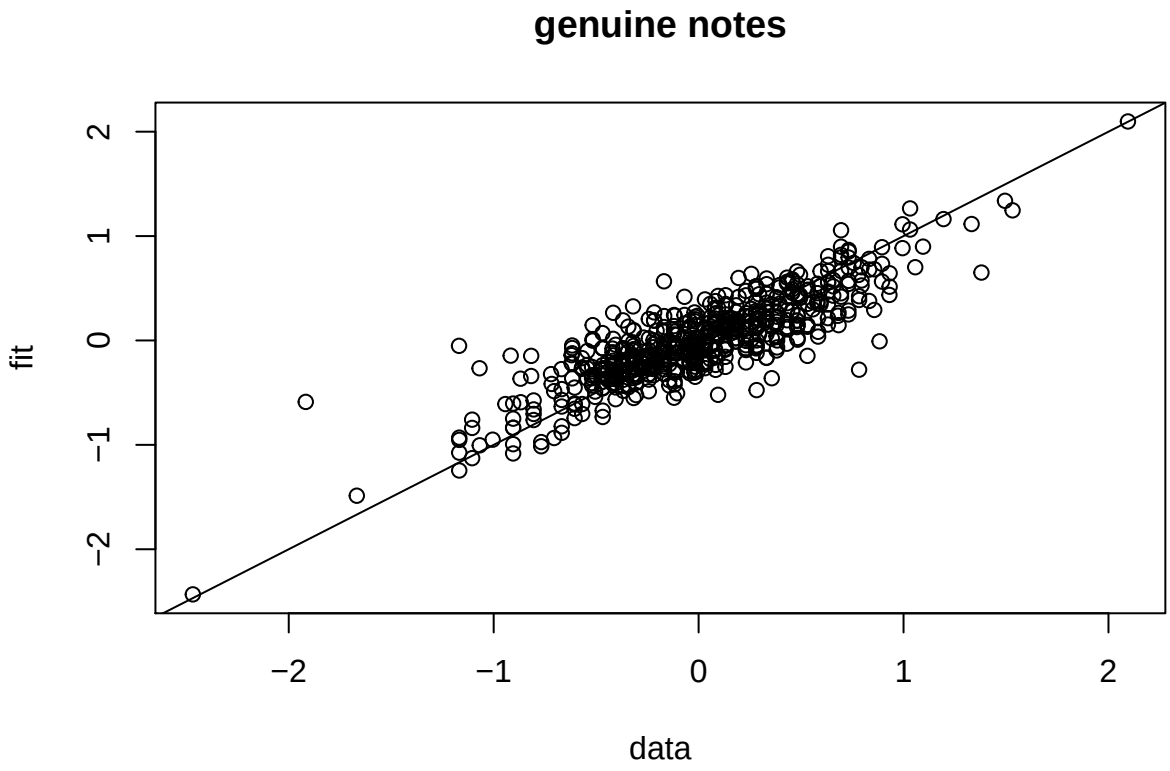
```
#calculate the one standard deviation
index = minIndex
#get index
while(crossv$crossv[index+1]< (crossv$crossv[minIndex] + crossv$sd.cv[minIndex]))
  index = index + 1
kk[index]
```

```
## [1] 0.041
```

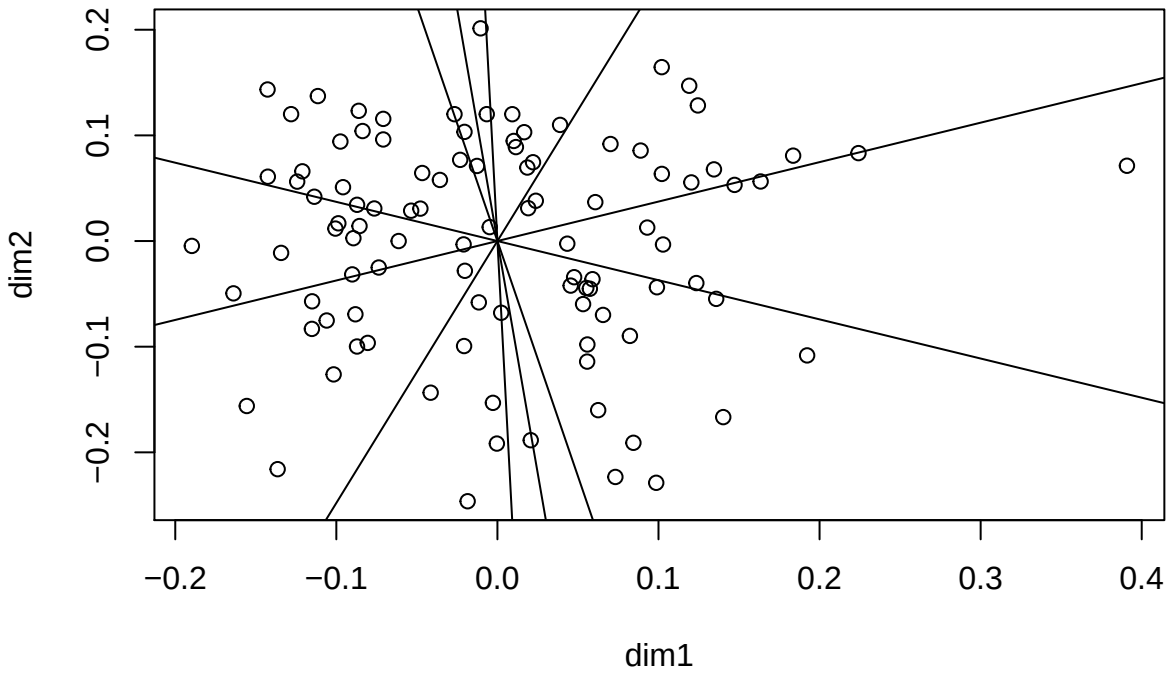
According to the ridege trace graph, lines are more stable after the point $\kappa = 0.041$. So for this graph, it performs better.\

5)

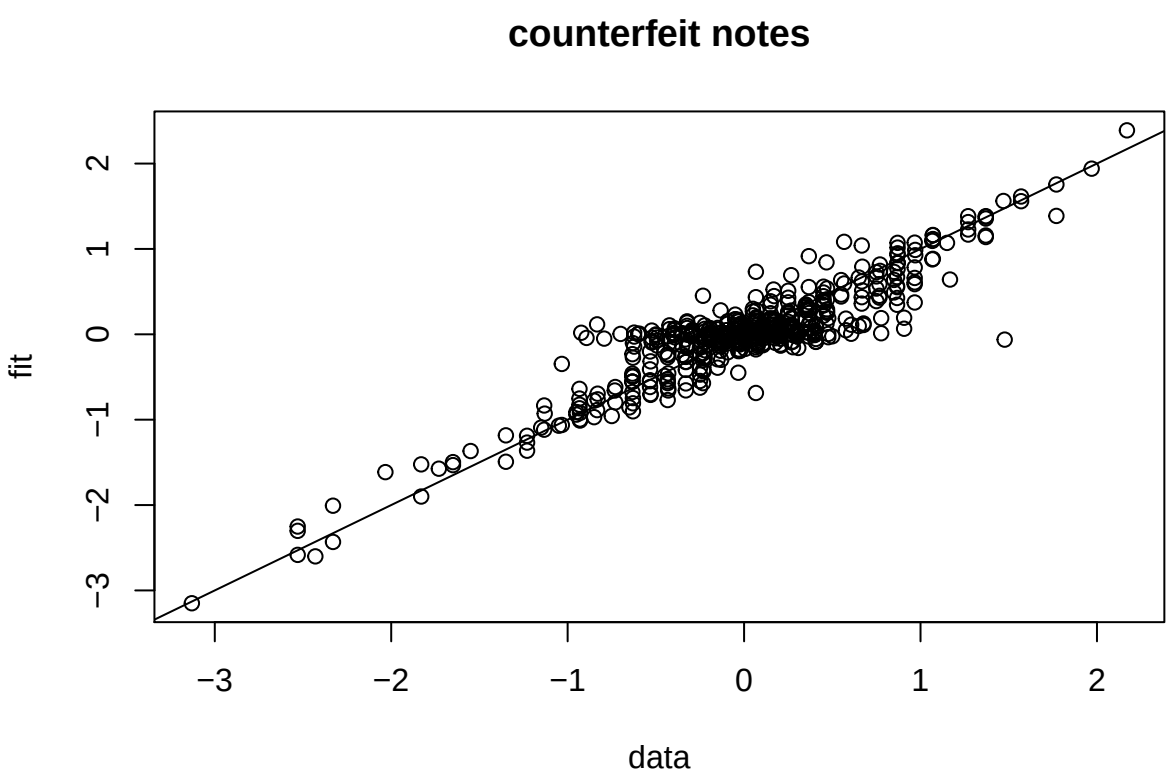
```r
data_bank = read.table("http://www.stat.ucla.edu/~handcock/202B/datasets/SwissBankNotes.txt", skip=22,h
data_bank = as.matrix(data_bank)
genuine = data_bank[1:100,]
counterfeit = data_bank[101:200,]
#genuine note
z = genuine
m<-colSums(z)/ nrow(z)
z<-z-outer(rep(0,nrow(z)),m,"+")
s<-svd(z,nu=2,nv=2)
u<-s$u; v<-s$v; d<-s$d; w<-v*outer(rep(1,ncol(z)),d[1:2])
f<-u%*%t(w)
plot(as.matrix(z),f,xlab="data",ylab="fit", main = "genuine notes")
abline(0,1)
```



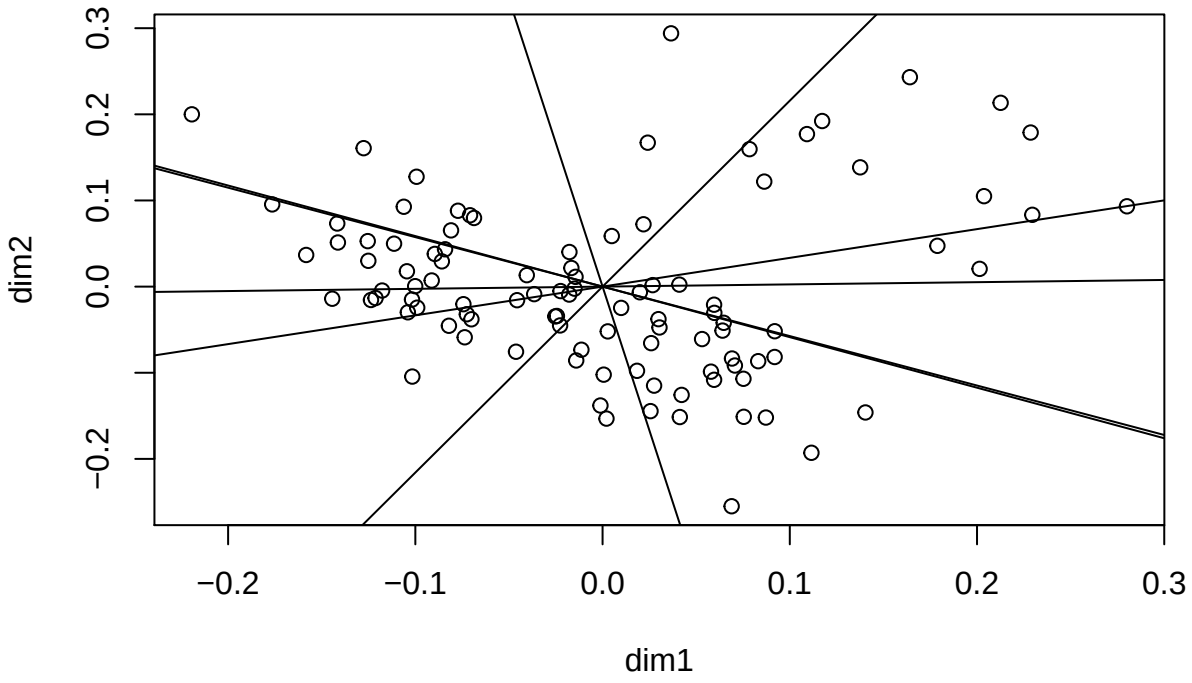

```r
plot(u,xlab="dim1",ylab="dim2")

for (i in 1:ncol(z))
  abline(0,w[i,2]/w[i,1])
```



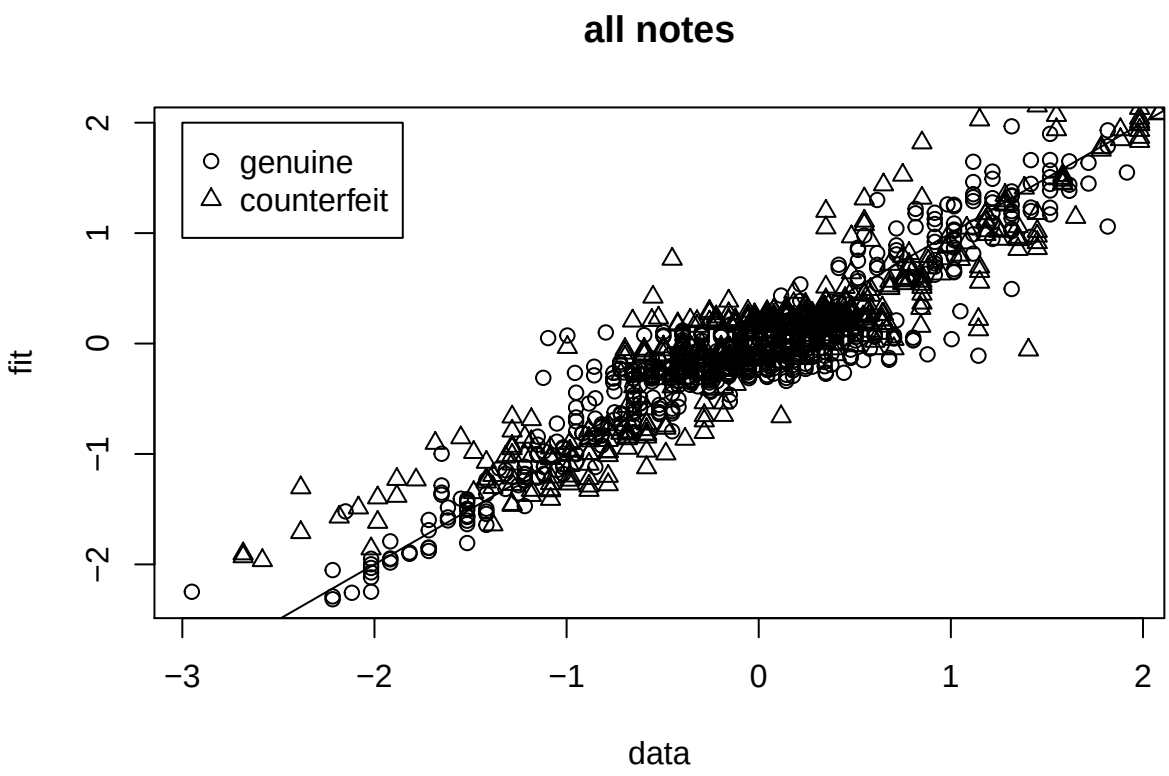

```
#counterfeit note
z = counterfeit
m<-colSums(z)/ nrow(z)
z<-z-outer(rep(0,nrow(z)),m,"+")
s<-svd(z,nu=2,nv=2)
u<-s$u; v<-s$v; d<-s$d; w<-v*outer(rep(1,ncol(z)),d[1:2])
f<-u%*%t(w)
plot(as.matrix(z),f,xlab="data",ylab="fit", main = "counterfeit notes")
abline(0,1)
```



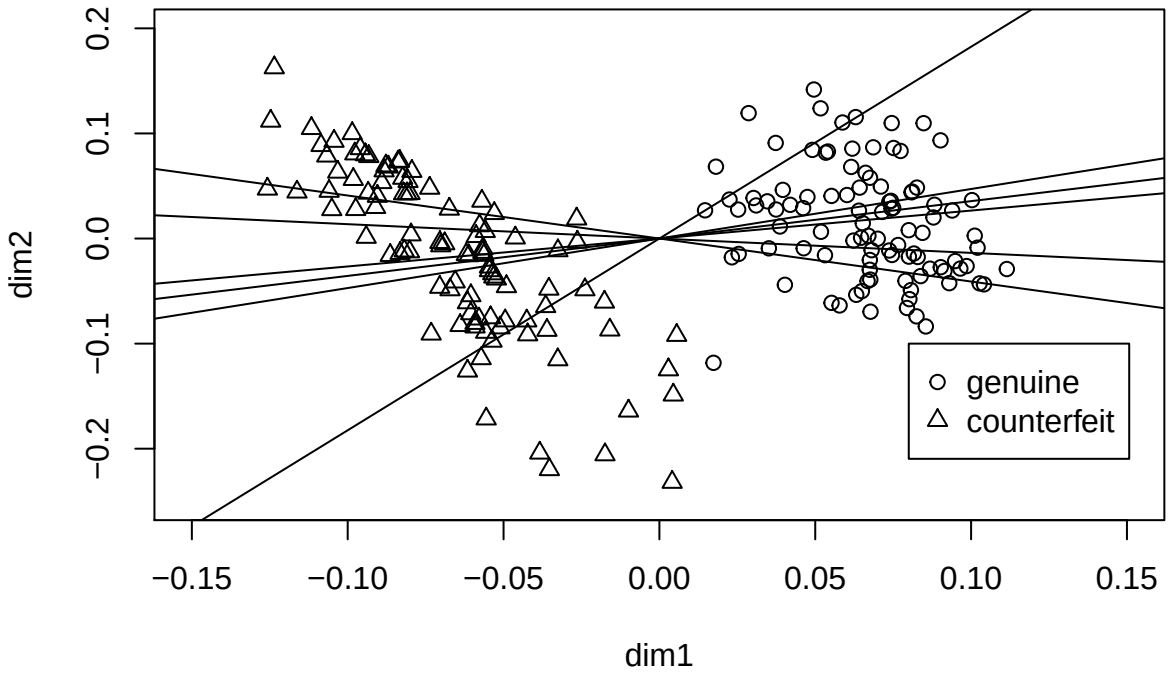

```
plot(u,xlab="dim1",ylab="dim2")
for (i in 1:ncol(z))
  abline(0,w[i,2]/w[i,1])
```




```
#all notes
z = data_bank
m<-colSums(z)/ nrow(z)
z<-z-outer(rep(0,nrow(z)),m,"+")
s<-svd(z,nu=2,nv=2)
u<-s$u; v<-s$v; d<-s$d; w<-v*outer(rep(1,ncol(z)),d[1:2])
f<-u%*%t(w)
plot(as.matrix(z[1:100,]),f[1:100,],xlab="data",ylab="fit", main = "all notes", pch = 1)
points(as.matrix(z[101:200,]),f[101:200,], pch = 2)
abline(0,1)
legend(x = -3, y = 2, legend = c("genuine", "counterfeit"),pch= c(1, 2))
```




```
plot(u[1:100,],xlab="dim1",ylab="dim2", pch = 1, xlim = c(-0.15, 0.15), ylim = c(-0.25, 0.2))
points(u[101:200,], pch = 2)
legend(x = 0.08, y = -0.1, legend = c("genuine", "counterfeit"),pch= c(1, 2))
for (i in 1:ncol(z))
  abline(0,w[i,2]/w[i,1])
```



Two-dimension fit of PCA performs quite well for all dimensions of data. From the fitted plot of all notes, counterfeit notes have more variation, which means the thrid dimension "explains" the data. We calculate the fitted values and plot the fiteed values against real values. From the graphs of all notes, the values of the first dimension of fitted counterfeit notes are likely negative while the values of genuine notes are positive.