Implemntation of SFPCA

Luofeng Liao

March 20, 2018

1 Introduction

The follow code implement the Rank-one SFPCA in *Sparse and FUnctional Principal Compoments Analysis, Gevenvera et al., 2013*. The challenge of proposing SFPCA lies in that 1) the model needs to take into account sparsity and smoothness of the principal components (PC), and 2) the optimiazation porblem should enjoy appropriate numerical property. In SFPCA algorithm, the original problem is divided into iterating solving penalized regression problem, which can be solved using proximal gradient methods or accelerated proximal gradient method.

2 Annotated code review

The `C` code will be posted after thorough testing.

```r
library('Rcpp')
library('RcppArmadillo')
setwd('C:/Users/Smart/Desktop')
sourceCpp('sfpca.cpp')
```

```r
# Some util functions
norm_vec <- function(x) sqrt(sum(x^2))
n <- 200
SSD <- function(n){
  # Generate squared second difference matrix, which looks like
  #   6   -4  1   0   0   0
  #   -4  6  -4   1   0   0
  #   1   -4  6   -4  1   0
  #   0   1   -4  6   -4  1
  #   ..
  #   I derived the formula and should be further verified
  a <- 6*diag(n)
  for(i in 1:n){
    for(j in 1:n){
      if(abs(i-j) == 1) a[i,j] = -4;
      if(abs(i-j) == 2) a[i,j] = 1;
    }
  }
  return(a);
}

uni <- function(n){
  # Noramlize a vector
  u_1 <- as.vector(rnorm(n))
  return(u_1/norm_vec(u_1))
}
```

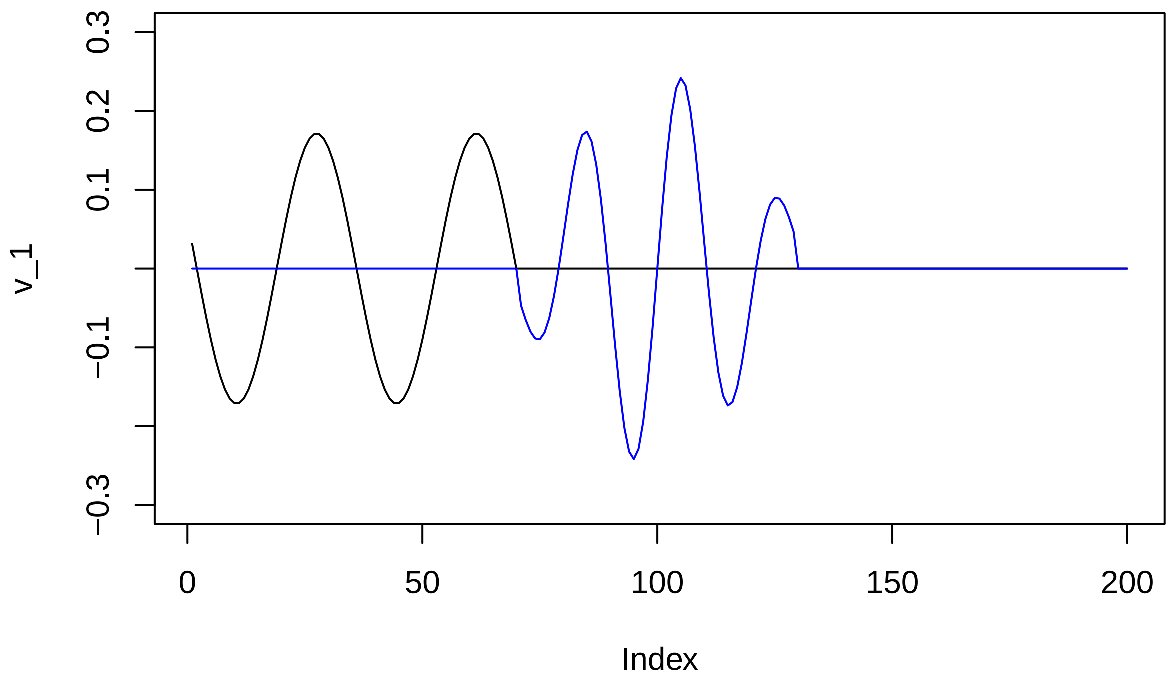
```r
# Create data, the same setting as section 5.1 Simulation Study in the paper
ind <- as.vector(seq(n))
u_1 <- uni(n)
u_2 <- uni(n)
u_3 <- uni(n)
eps <- matrix(rnorm(n*n),n,n)
eps <- eps/20

# Sinusoidal
v_1 <- sin((ind+15)*pi/17);v_1[floor(7/20*n):n]=0;v_1 <- v_1/norm_vec(v_1);

# Gaussian-modulated sinusoidal
v_2 <- as.vector(exp(-(ind-100)^2/650)*sin((ind-100)*2*pi/21));
v_2[0:floor(7/20*n)]=0;
v_2[floor(130/200*n):n] = 0;
v_2 <- v_2/norm_vec(v_2);

# Sinusoidal
v_3 <- sin((ind-40)*pi/30);v_3[0:floor(130/200*n)]=0;v_3 <- v_3/norm_vec(v_3);

plot(v_1,type = 'l',ylim=c(-0.3,0.3));
lines(v_2,col='blue');
```



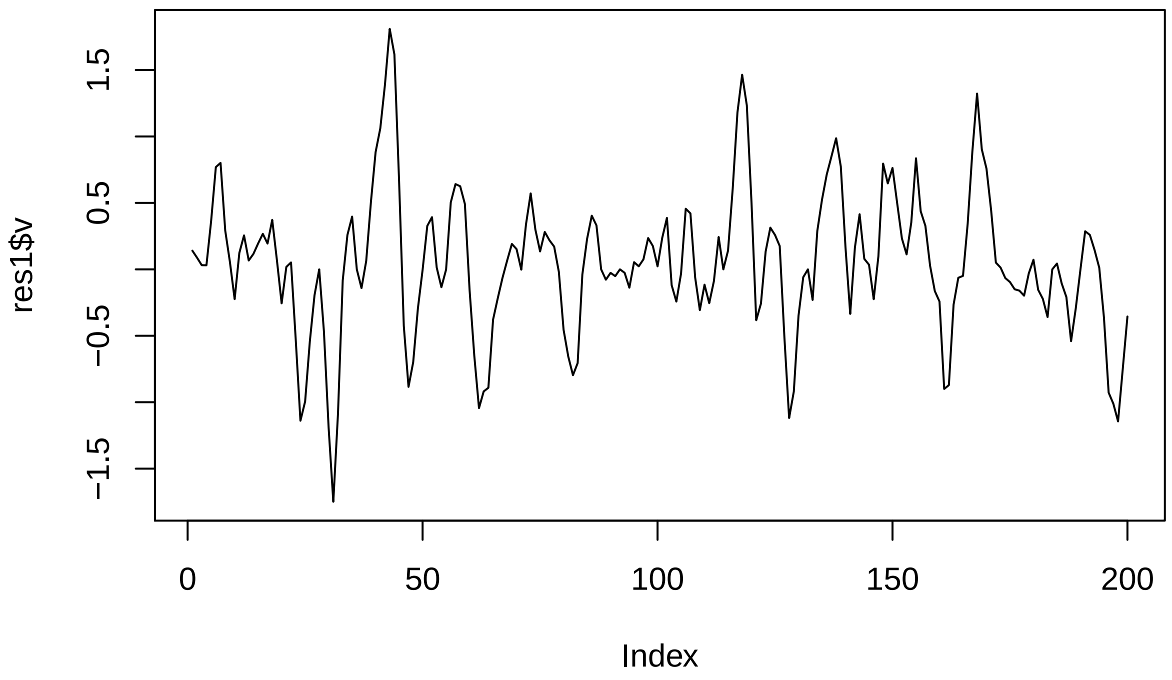
```r
#lines(v_3,col='red')
```

```r
X <-  n/4*u_1 %*% t(v_1) + n/5*u_2 %*% t(v_2) +eps
print(norm(X) /norm(eps))
```

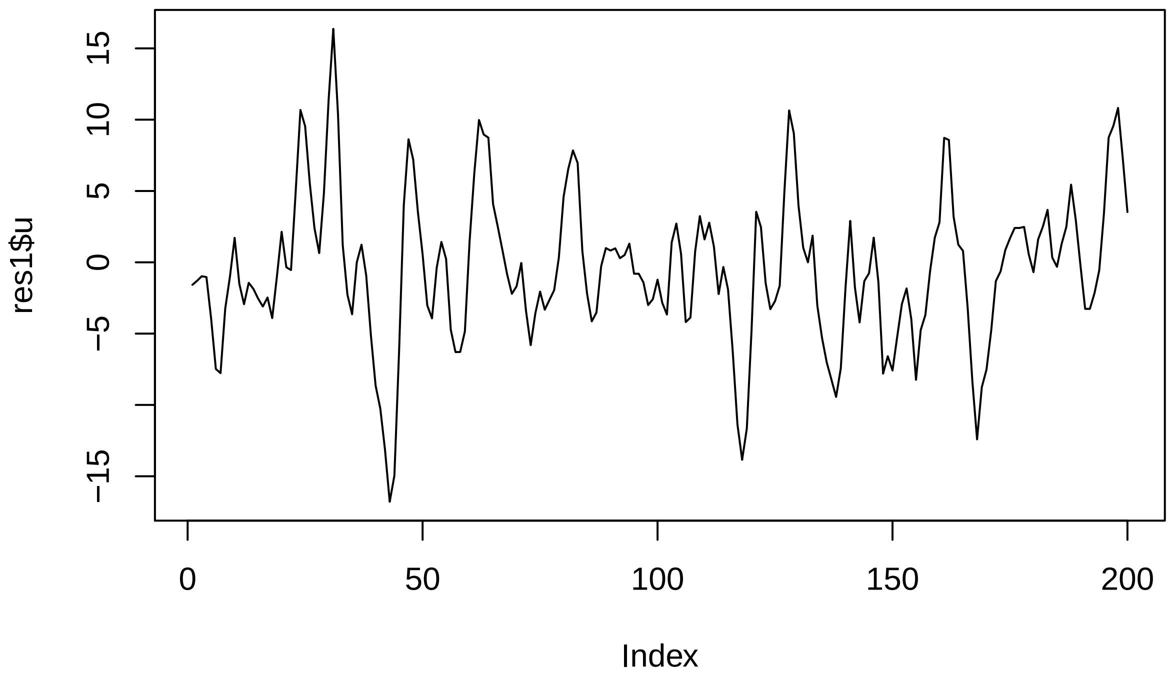
```
## [1] 12.40707
```

```r
# --------------------
# L1 penelty for sparsity
P_u <- "l1"
P_v <- "l2"
#---------------------
# Squared second difference matrix for smoothness
O_u <- SSD(n)
O_v <- O_u
#----------------------
# Smoothness
a_u <- 1
a_v <- 1
#----------------------
# Sparsity
lam_u <- 0.1
lam_v <- 0.1
```

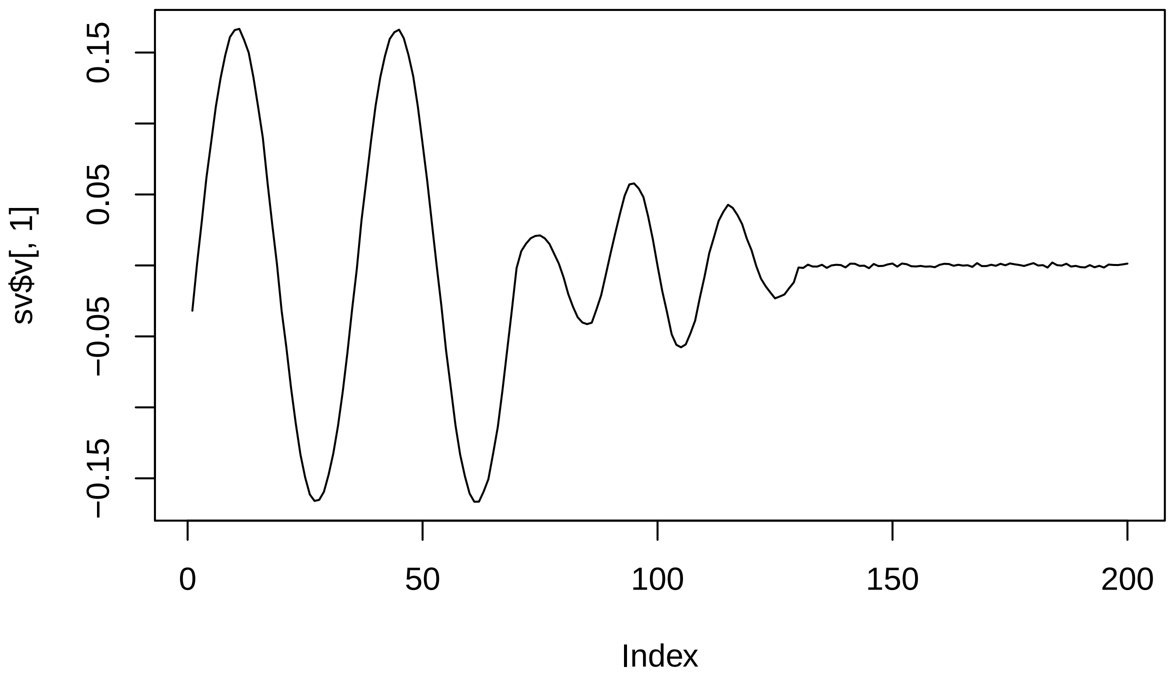
```r
# Run the model
# Input:
# ---- X: data matrix
# ---- O_u, O_v: Squared second difference matrix, lambda_u and lambda_v control the penalty level
# ---- "l1", "l1": proximal operator has closed form solution of L1.
# Output:
# ---- u,v: the PCs
# ---- uTXv
res1 <- sfpca(X,O_u,O_v,a_u,a_v,lambda_u=lam_u,lambda_v=lam_v,"l1","l1",1e-9,1e+5)
plot(res1$v,type='l')
```



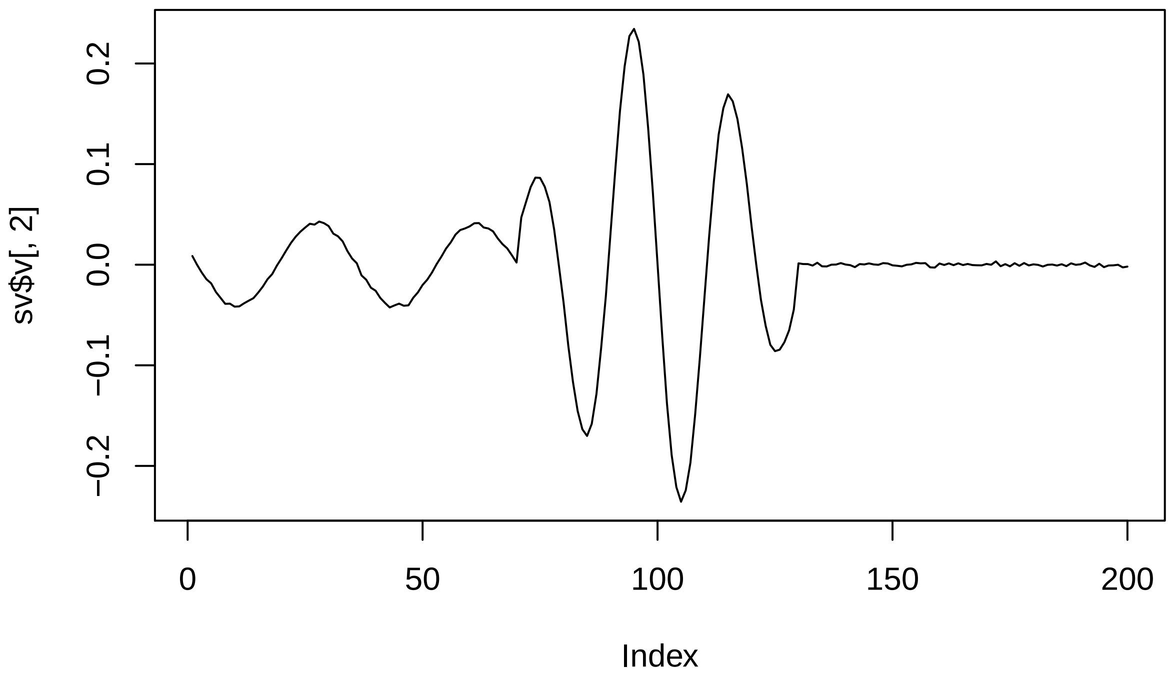
```r
plot(res1$u,type='l')
```



```
sv <- svd(X)
plot(sv$v[,1],type='l')
```



```r
plot(sv$v[,2],type='l')
```



We can see from the result, the implentation is questionable. Although smoothing penalty is working, is far from what is desired.

By contrast, SVD emits better results. The signal from v_1 is successfully extracted, with little remaining $v2$ signal, which has much lower climax than the original signal due to the L_1 penalty. More sparse results can be abtained by choosing appropriate penalizing constant.

3 TODO List

3.0 Fix the bug. Double check everything.

3.1 Test the degenerated cases, i.e., when $\alpha_u, \alpha_v, \Omega_u, \Omega_v$ are zeros, the algorithm should be the SVD.

3.2 Try to use deflation method to estimate rank 2 and rank 3 models. Try to attain the same result as the paper.