PCA for timeOmics data

PCA timeOmics Package with timeOmics data

```
###Lastest Bioconductor Release
## install BiocManager if not installed
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager",repos = "http://cran.us.r-project.org")


## install timeOmics
BiocManager::install('timeOmics')

##Lastest Github version
install.packages("devtools",repos="http://apt.sw.be/redhat/el7/en/$ARCH/extras")
# then load
library(devtools)
install_github("abodein/timeOmics")

#Load the package
library(timeOmics)
##Useful package to run this vignette
library(tidyverse)
library(conflicted)
library(writexl)
library(readxl)
```

Running PCA without any filtering

Note: after running this code you can see that there is no difference between the result of mixOmics::PCA and stats::prcomp

```
data("timeOmics.simdata")
sim.data <- timeOmics.simdata$sim

dim(sim.data)
```

```
[1] 45 21
```

```
head(sim.data[,1:7])
```

```
           c0       c1.0       c1.1        c1.2      c1.3        c1.4
A_1 0.6810022 -0.1681427 -0.1336986  0.12040677 0.4460119 -0.93382470
A_2 1.4789556  0.4309468  1.1172245 -0.08183742 0.4585589 -0.56857351
A_3 0.9451049  1.4676125  1.6079441 -0.11034711 1.5761445 -0.09178880
A_4 0.7403461  1.1211525  1.7702314  0.17460753 1.4079393 -0.00414130
A_5 0.9291161  1.2387863  1.8332048 -0.03780133 1.2714786  0.01158791
A_6 1.0408472  2.3145195  2.5332477  0.23133263 2.1085377  0.81762482
          c2.0
A_1 -0.3369326
A_2 -0.6208655
A_3 -1.1399966
A_4 -0.8660105
A_5 -1.2250107
A_6 -1.7240044
```

```
pca.res <- pca(X = sim.data, ncomp = 5, scale=T, center=T)
pca.res[["cum.var"]]
```

```
      PC1       PC2       PC3       PC4       PC5
0.5878299 0.9271460 0.9744594 0.9830301 0.9883113
```

```
pca.res[["rotation"]]
```

```
              PC1         PC2          PC3          PC4          PC5
c0    0.008171982  0.05115184 -0.992935864  0.058847963 -0.007486662
c1.0 -0.224507281  0.22675907  0.027514486 -0.039903518 -0.024074105
c1.1 -0.222863834  0.23079837  0.015133025  0.001363713  0.036056657
c1.2 -0.222288939  0.20738469 -0.017972987 -0.023972556  0.803942943
```

```
c1.3 -0.220950544  0.23091024  0.032416886  0.026968529 -0.178260257
c1.4 -0.224572414  0.22800681  0.021191688  0.026038521 -0.015835816
c2.0  0.223645058 -0.22769776 -0.008006748  0.061630205  0.014821792
c2.1  0.226778948 -0.22454754 -0.024015584 -0.003717985 -0.048782944
c2.2  0.232843327 -0.19133404  0.022100774 -0.012105940  0.473365368
c2.3  0.227506017 -0.22109334  0.017945486 -0.043383465  0.108238606
c2.4  0.222170312 -0.23140365 -0.024701105 -0.039108027  0.064779199
c3.0  0.221741593  0.23160353  0.017272654 -0.028334638 -0.131553473
c3.1  0.225123161  0.22780585  0.009090558 -0.006665060  0.015273980
c3.2  0.217289749  0.20888876  0.023049852  0.675747041  0.040328673
c3.3  0.220332433  0.23216264  0.016612667  0.019264871  0.207744339
c3.4  0.226952246  0.22220850  0.017822594  0.018864206  0.000206243
c4.0 -0.223309822 -0.22894832 -0.002060185 -0.095533547  0.111415543
c4.1 -0.227598537 -0.22316173 -0.027010740 -0.021894948 -0.020772991
c4.2 -0.207737782 -0.22256689  0.063548958  0.714844239  0.010584867
c4.3 -0.228392299 -0.21620967 -0.044336788  0.082232220  0.042943174
c4.4 -0.224303287 -0.22831320 -0.031046883  0.003708396 -0.001233630
```

Data preprocessing

In a longitudinal context, one can be interested only in features that vary over time and filter out molecules with a low variation coefficient.

To do so, we can first naively set a threshold on the variation coefficient and keep those features that exceed the threshold.

Note: After running this code you will see the changes in dimension (feature C0 is filtered out)

```r
remove.low.cv <- function(X, cutoff = 0.5){
  # var.coef
  cv <- unlist(lapply(as.data.frame(X),
                      function(x) abs(sd(x)/mean(x))))
  return(X[,cv > cutoff])
}

data.filtered <- remove.low.cv(sim.data, 0.5)
dim(data.filtered)
```

```
[1] 45 20
```

```r
head(data.filtered[,1:7])
```

```
          c1.0       c1.1        c1.2      c1.3        c1.4       c2.0
A_1 -0.1681427 -0.1336986  0.12040677 0.4460119 -0.93382470 -0.3369326
A_2  0.4309468  1.1172245 -0.08183742 0.4585589 -0.56857351 -0.6208655
A_3  1.4676125  1.6079441 -0.11034711 1.5761445 -0.09178880 -1.1399966
A_4  1.1211525  1.7702314  0.17460753 1.4079393 -0.00414130 -0.8660105
A_5  1.2387863  1.8332048 -0.03780133 1.2714786  0.01158791 -1.2250107
A_6  2.3145195  2.5332477  0.23133263 2.1085377  0.81762482 -1.7240044
          c2.1
A_1 -0.0677313
A_2 -0.2950588
A_3 -1.9646258
A_4 -1.4746159
A_5 -1.7628437
A_6 -2.6352175
```

Time Modelling

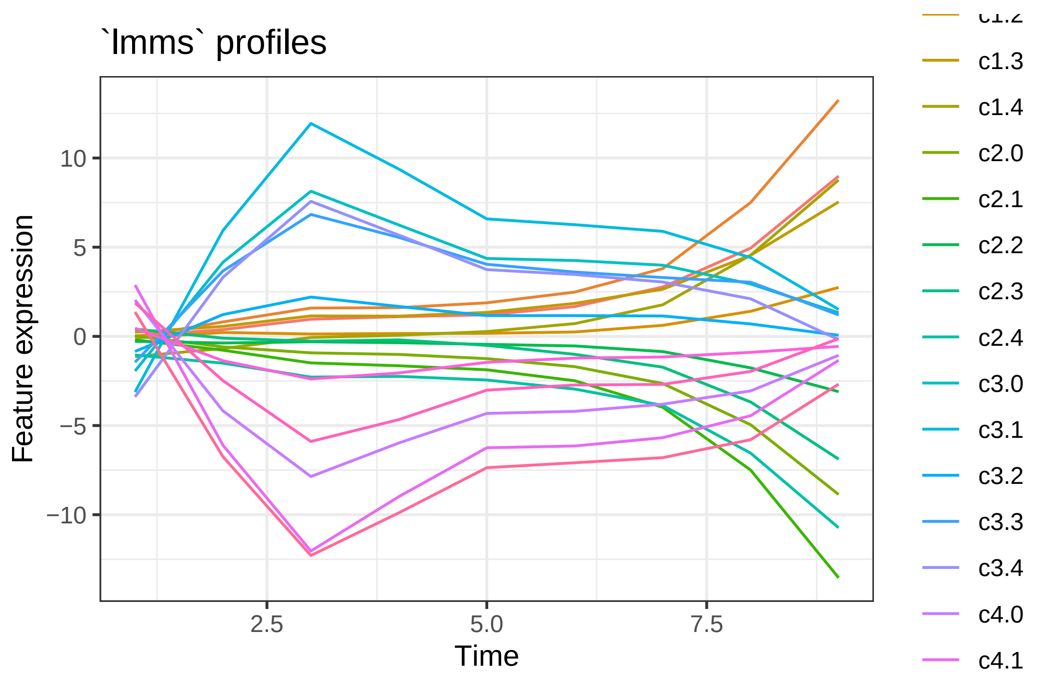
The next step is the modelling of each feature (molecule) as a function of time.

It fit the data of each feature based on different time points to a spline.

```r
devtools::install_github("cran/lmms", force = TRUE)
library(lmms)

# numeric vector containing the sample time point information
time <- timeOmics.simdata$time
head(time)

# example of lmms
lmms.output <- lmms::lmmSpline(data = data.filtered, time = time,
                         sampleID = rownames(data.filtered), deri = FALSE,
                         basis = "p-spline", numCores = 4, timePredict = 1:9,
                         keepModels = TRUE)
modelled.data <- t(slot(lmms.output, 'predSpline'))
```

Let's plot the modeled profiles.

```
# gather data
data.gathered <- modelled.data %>% as.data.frame() %>%
  rownames_to_column("time") %>%
  mutate(time = as.numeric(time)) %>%
  pivot_longer(names_to="feature", values_to = 'value', -time)

# plot profiles
ggplot(data.gathered, aes(x = time, y = value, color = feature)) + geom_line() +
  theme_bw() + ggtitle("`lmms` profiles") + ylab("Feature expression") +
  xlab("Time")
```



Profile filtering

for removing noisy profile

use the filtered data for PCA

Note: After running this code, there are no changes in dimension of data

```r
filter.res <- lmms.filter.lines(data = data.filtered,
                                lmms.obj = lmms.output, time = time)
profile.filtered <- filter.res$filtered
dim(profile.filtered)
```

```
[1] 45 20
```

```r
head(profile.filtered[,1:7])
```

```
          c1.0       c1.1        c1.2      c1.3        c1.4       c2.0
A_1 -0.1681427 -0.1336986  0.12040677 0.4460119 -0.93382470 -0.3369326
A_2  0.4309468  1.1172245 -0.08183742 0.4585589 -0.56857351 -0.6208655
A_3  1.4676125  1.6079441 -0.11034711 1.5761445 -0.09178880 -1.1399966
A_4  1.1211525  1.7702314  0.17460753 1.4079393 -0.00414130 -0.8660105
A_5  1.2387863  1.8332048 -0.03780133 1.2714786  0.01158791 -1.2250107
A_6  2.3145195  2.5332477  0.23133263 2.1085377  0.81762482 -1.7240044
          c2.1
A_1 -0.0677313
A_2 -0.2950588
A_3 -1.9646258
A_4 -1.4746159
A_5 -1.7628437
A_6 -2.6352175
```

Single-Omic longitudinal clustering

Note: After conducting timeOmics::PCA you can see the differences because of filtering out the C0 feature

```r
# run pca
pca.res <- pca(X = profile.filtered, ncomp = 5, scale=T, center=T)
pca.res[["cum.var"]]
```

```
      PC1       PC2       PC3       PC4       PC5 
0.6171836 0.9726619 0.9818106 0.9873586 0.9909647 
```

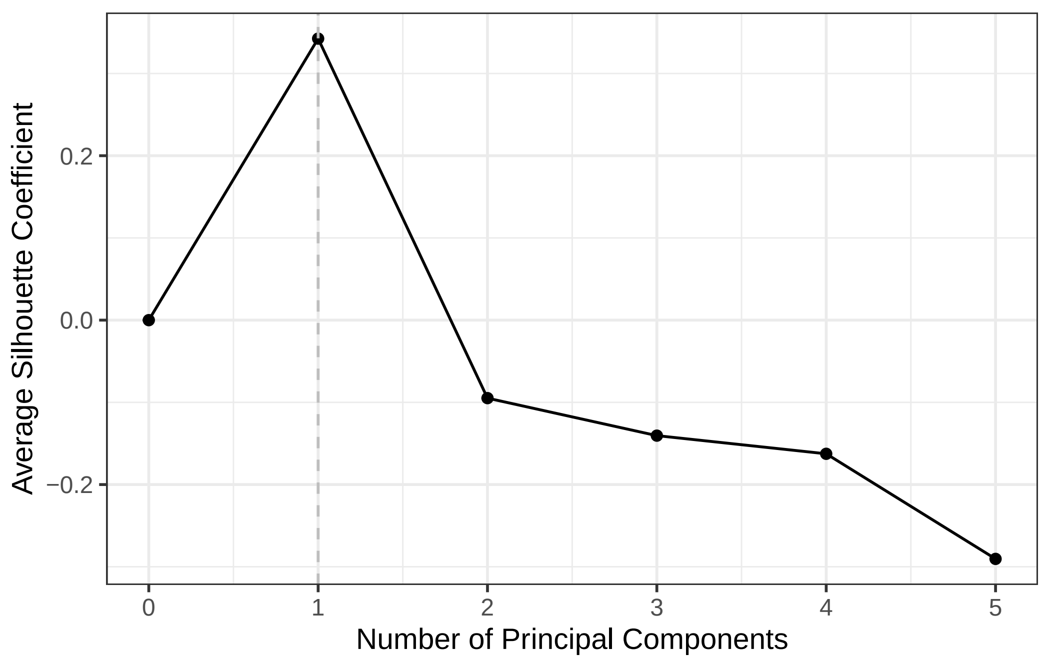
```r
pca.res[["rotation"]]
```

```
            PC1        PC2          PC3           PC4         PC5
c1.0  0.2246145 -0.2269778  0.031178119 -0.0267093197  0.18634956
c1.1  0.2229811 -0.2309341 -0.004085274  0.0351484549  0.11830279
c1.2  0.2224141 -0.2072779  0.037674815  0.8070537171 -0.36138521
c1.3  0.2210574 -0.2311629 -0.037489595 -0.1810636827  0.25030141
c1.4  0.2246846 -0.2281790 -0.030564396 -0.0167806336  0.04227275
c2.0 -0.2237651  0.2277822 -0.059648026  0.0159660043 -0.16398884
c2.1 -0.2268876  0.2247347  0.010141690 -0.0469388992 -0.10684110
c2.2 -0.2329631  0.1911731 -0.002784402  0.4666361840  0.78580153
c2.3 -0.2276403  0.2209768  0.035016907  0.1064842294 -0.10306169
c2.4 -0.2222821  0.2316043  0.045044523  0.0661207202 -0.11928510
c3.0 -0.2216380 -0.2321377  0.026118978 -0.1317100184 -0.06846916
c3.1 -0.2250158 -0.2282760  0.006436224  0.0148944189  0.04374798
c3.2 -0.2171979 -0.2093994 -0.663980980  0.0478923665 -0.10237588
c3.3 -0.2202277 -0.2326871 -0.020152231  0.2075111494 -0.06545576
c3.4 -0.2268534 -0.2227346 -0.021882171 -0.0006509649  0.01705890
c4.0  0.2231968  0.2293632  0.090907087  0.1096353013  0.05720175
c4.1  0.2275055  0.2237578  0.026669466 -0.0199785050  0.02734853
c4.2  0.2075911  0.2225596 -0.729700203  0.0105111176  0.08901167
c4.3  0.2283139  0.2169209 -0.066309668  0.0477205466 -0.19942075
c4.4  0.2242100  0.2289392  0.003805102  0.0006166509 -0.04721550
```

```r
# tuning ncomp
pca.ncomp <- getNcomp(pca.res, max.ncomp = 5, X = profile.filtered,
                      scale = T, center=T)

pca.ncomp$choice.ncomp
```

```
[1] 1
```

```r
#plot
plot(pca.ncomp)
```



```r
# final model
pca.res <- pca(X = profile.filtered, ncomp = 2, scale = FALSE, center=FALSE)

# extract cluster
pca.cluster <- getCluster(pca.res)
head(pca.cluster)
```

```
  molecule comp contrib.max cluster block contribution
1     c1.0  PC2 -0.27080548      -2     X     negative
2     c1.1  PC2 -0.39683076      -2     X     negative
3     c1.2  PC2 -0.08571738      -2     X     negative
4     c1.3  PC2 -0.22259184      -2     X     negative
5     c1.4  PC2 -0.28978510      -2     X     negative
6     c2.0  PC2  0.26703159       2     X     positive
```