MOSS: Multi-omic integration via sparse singular value decomposition

Working example of MOSS' application on pan-cancer multi-omic data.

Agustin Gonzalez-Reymundez

April 23, 2020

Contact: gonza@msu.edu

Contents

Introduction.

The following codes provide an example of application of MOSS for omic integration of a large data consisting of 33 cancer types across 5008 samples and and 60112 features (expression of 20319 genes -**GE**-, methylation at 28241 CpG islands -**METH**-, and copy number variant intensity for 11552 genes -**CNV**-). from The Cancer Genome Atlas (TCGA) data set (Chang et al., 2013). For a description of the data edition and quality controls, please refer to Gonzalez-Reymundez and Vazquez (2020). The whole example takes ~53 minutes on a Dell desktop computer (XPS 8900, x64, 06B8, Intel i7-67000 CPU), running under Windows 10 (see session info at the end of the document).

Retrieving and loading pan-cancer data from repository.

The following lines load MOSS and additional packages for visualizing results and handling FBM matrices.

```
require(bigstatsr)
require(ggplot2)
require(ggpmisc)
require(Rtsne)
library(MOSS)
```

The following code assumes you have the file "tcga_pan-cancer.rda" saved in your current folder. Then package bigstastr is used to turn each block into a Filed-backed Big Matrix (FBM).

```
load("tcga_pancancer.rda")
omic_blocks <- lapply(tcga_pancancer$omic_blocks, bigstatsr::as_FBM)
```

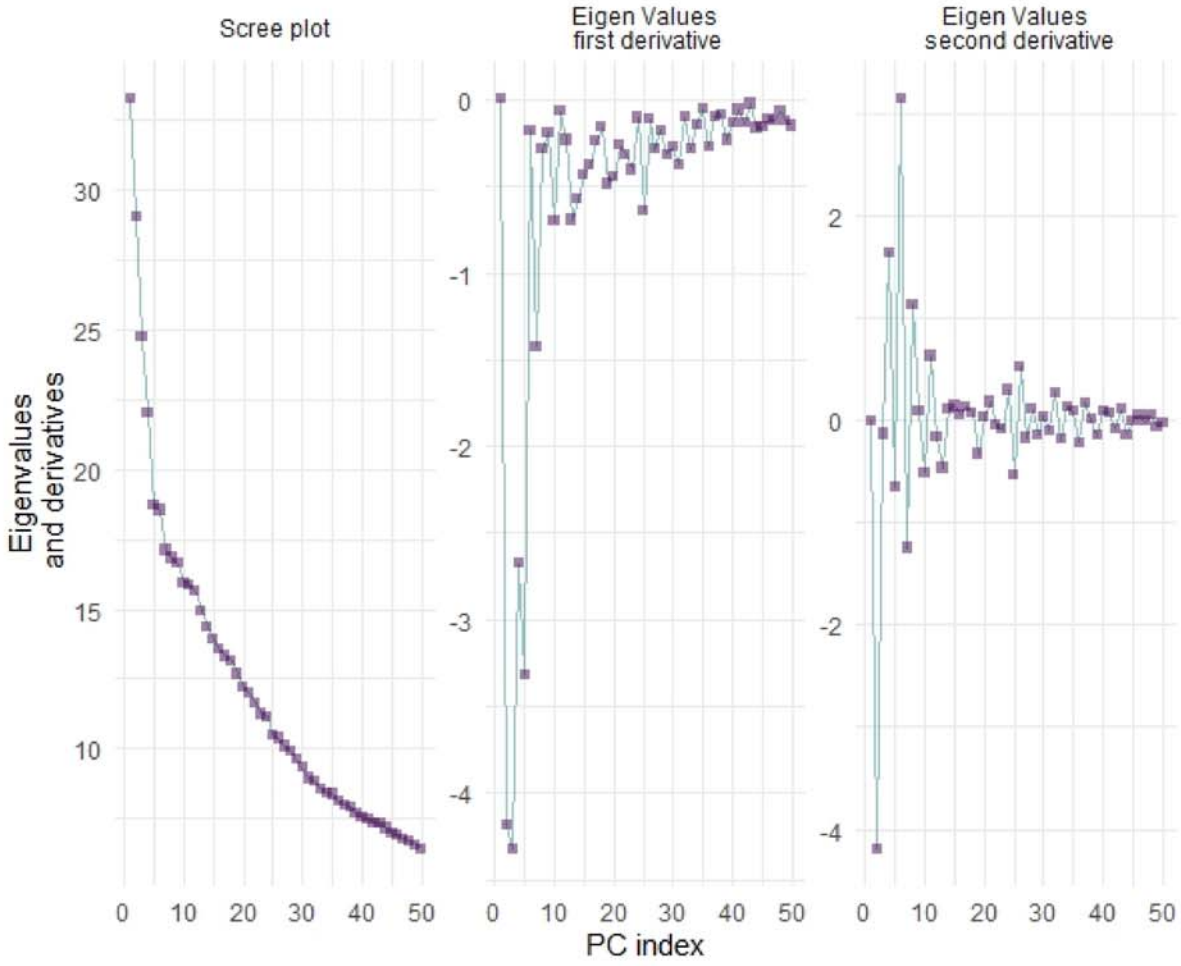
Omic integration of pan-cancer omic data via sparse principal components.

The following code shows how to run a sparse principal components analysis, by specifying a grid of degrees of sparsity from 1 to 100. Notice that since a elastic net type of penalization is being used, the degrees of sparsity do not directly correspond to number of features. For that reason, some eigenvectors will have a number of non-zero elements potentially larger than the selected degrees of sparsity.

```
set.seed(347)
out <- moss(data.blocks = omic_blocks,
            method = 'pca',
            scale. = TRUE,
            norm. = TRUE,
            K.X = 50,
            tSNE = list(perp=100,n.iter=1e3,n.samples=1),
            clus.lab = tcga_pancancer$tcga_metdat$organ_type,
            dg.grid.right = 1:100,
            alpha.right = 0.5,
            plot = TRUE)
```

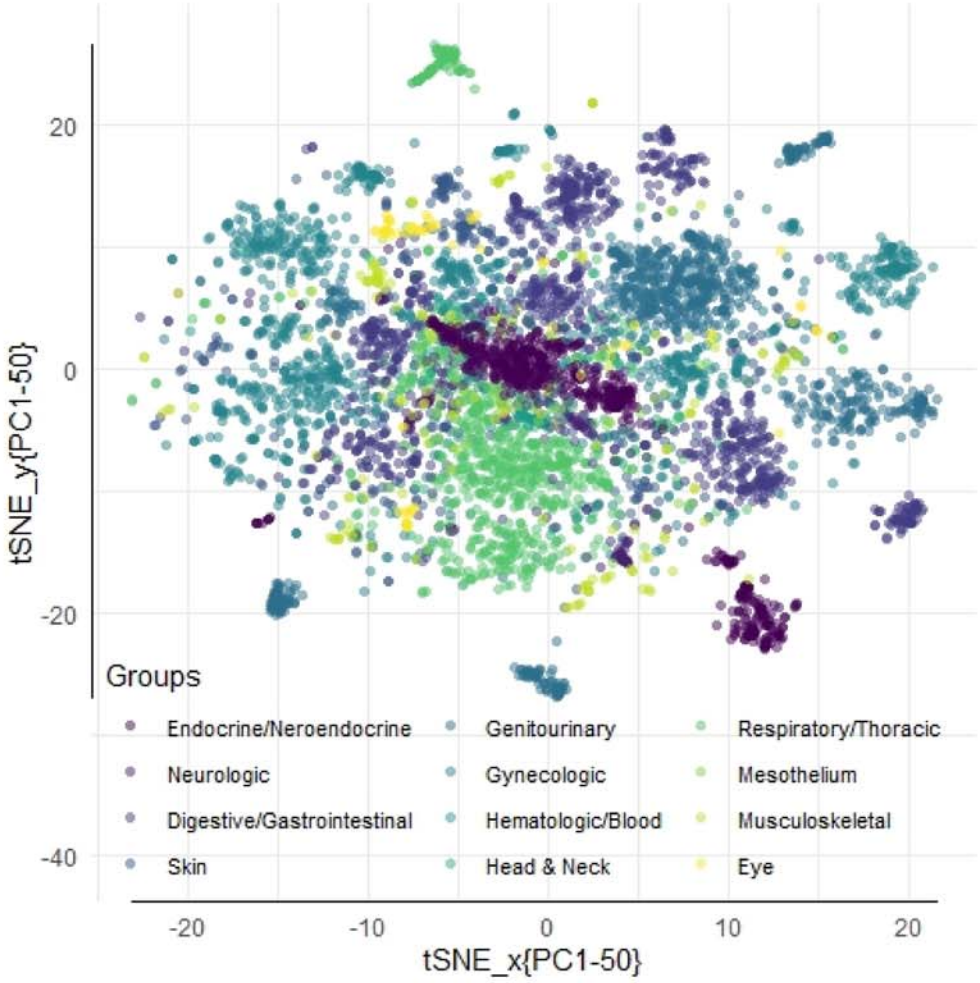
The following code returns a plot of eigenvalues, together with their first and second derivative across PC index:

```
#Showing scree plot.
out$scree.plot
```



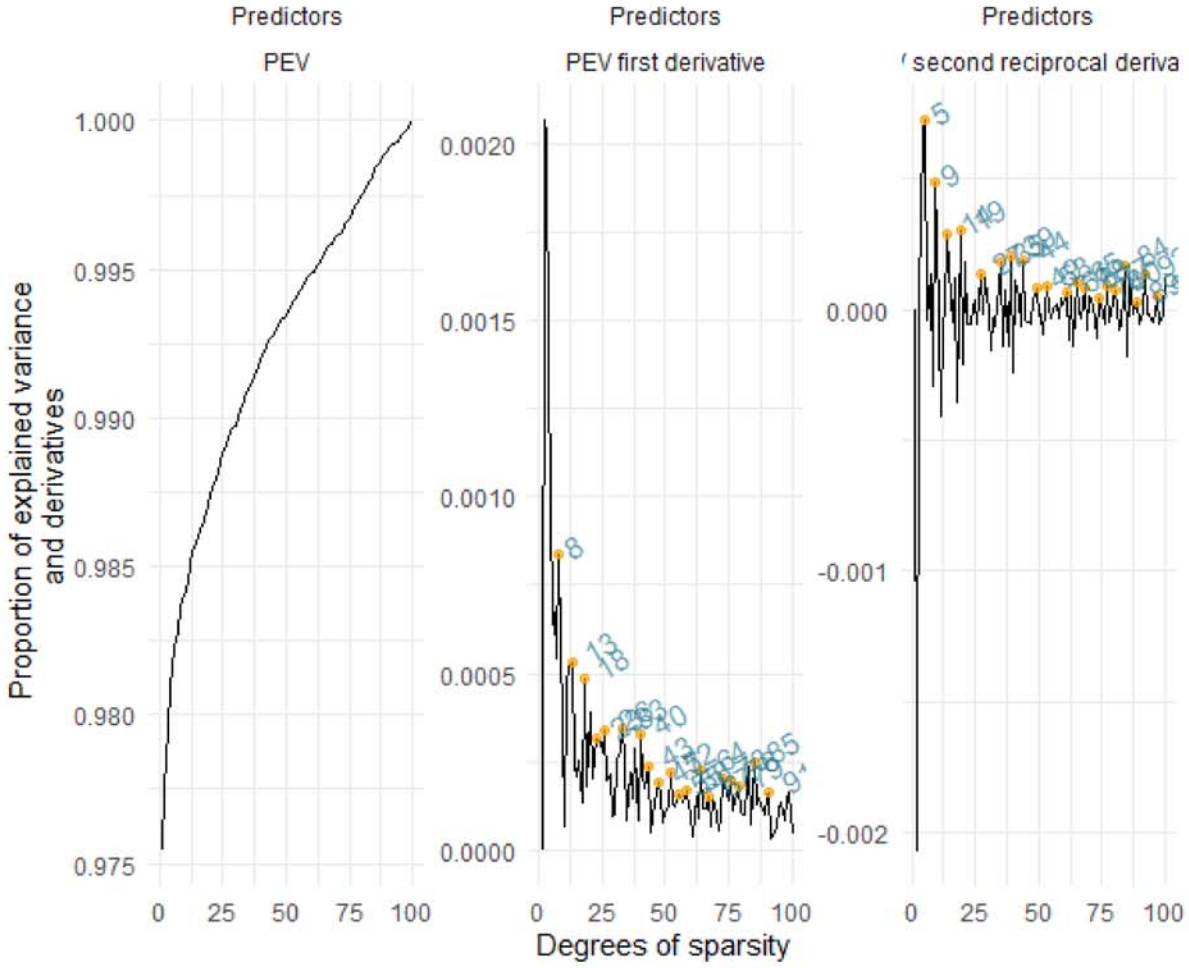
A tSNE map:

```
out$tSNE.plot
```



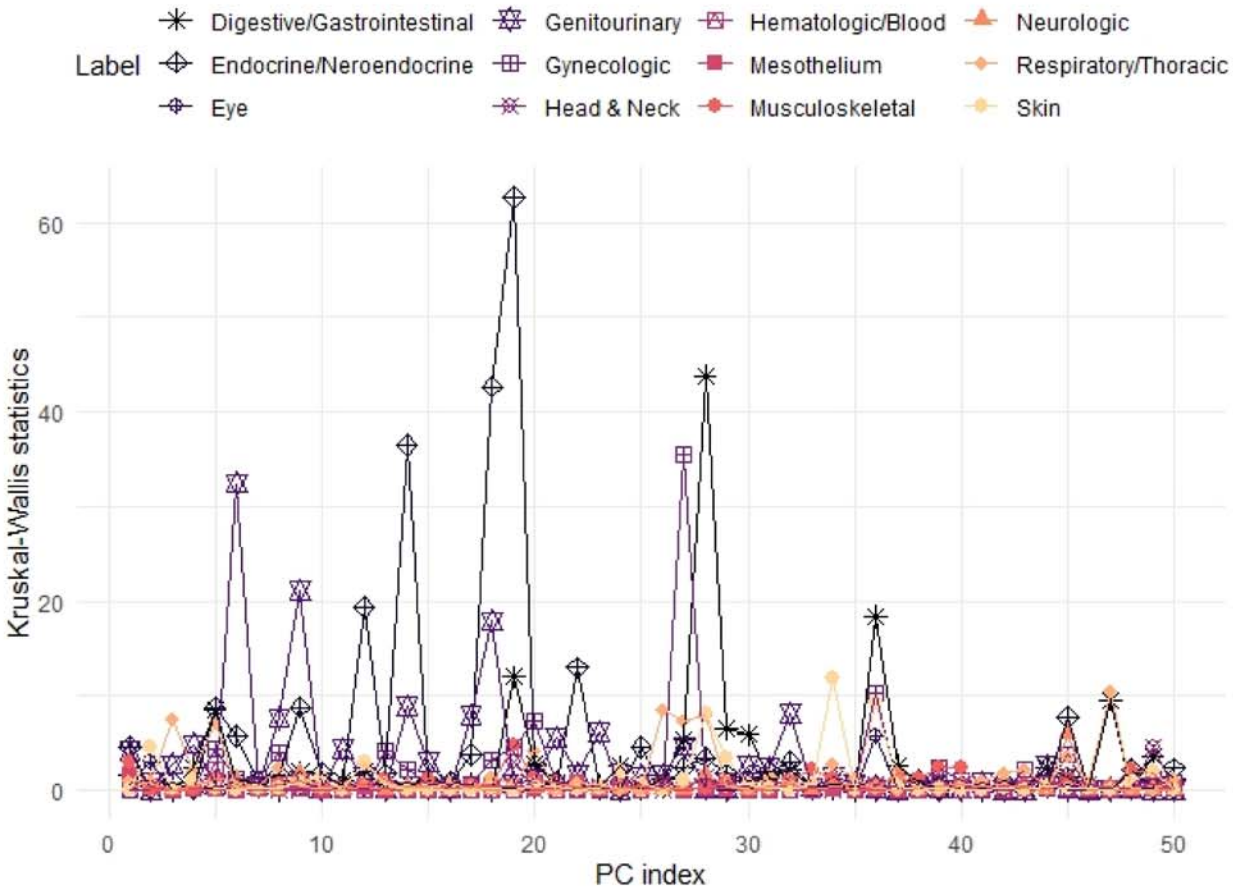
A plot with the marginal variation in *PEV* across degrees of sparsity at features levels. The numbers in the peaks of first and second-reciprocal derivatives show the point of rapid and/or accelerated changes. The minimum between the global maximum in the trajectory of first and second derivative, respectively, is automatically chosen as 'optimal':

```
out$tun_dgSpar.plot
```



A plot showing the association between elements in *clus.lab* and each PC. In this example, we used a broad classification of tumors by organ type. Classification of tumors by cancer type can be also addressed by specifying $clus.lab = tcga\_pancancertcga_metdatcancer\_type$. The association between organ types and PC is addressed with the KW statistics (higher values represent higher association):

```
#What axes of variation are associated with clusters of subjects?
out$subLabels_vs_cluster
```



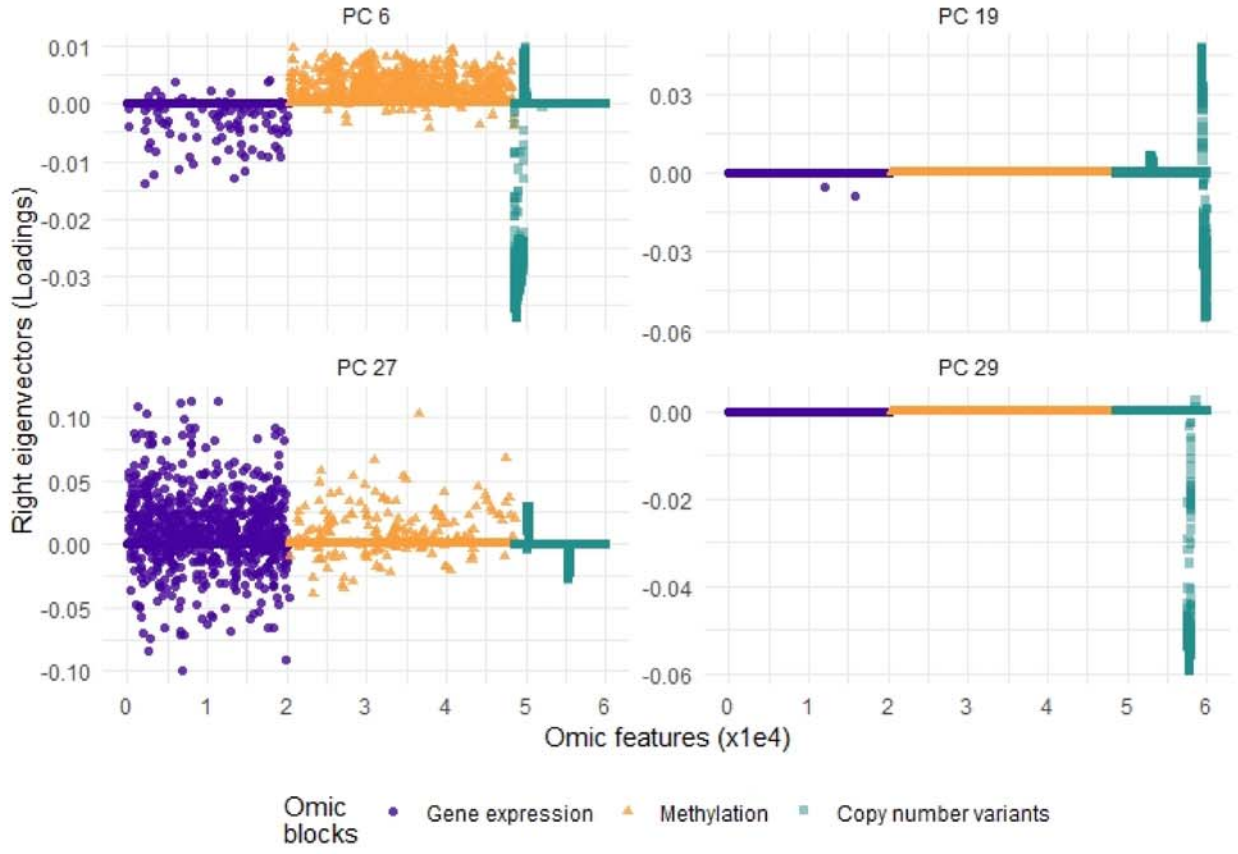
Finally, a plot of the selected features can be obtained by creating data frames with the desired PCs. Here, we have used PC6, 19, 27 and 29 based on their high KW values. The following code shows how to plot this results using **ggplot2**.

```
p <- Reduce(f = "+",
            x = lapply(X = omic_blocks,
                       FUN =  ncol))
d <- data.frame(
  Features=rep(x = 1 : p,
               times = 4),
  Loadings=as.vector(x = out$sparse$v[, c(6, 19, 27, 29)]),
  Features_groups=rep(x = rep(x = names(omic_blocks),
                              times=lapply(X = omic_blocks,
                                           FUN = ncol)),
                      times=4),
  PC=paste0("PC ",
            rep(x = c(6,19,27,29),
                each=p)))

d$PC <- factor(x = d$PC,
               levels = paste0("PC ",
                               c(6,19,27,29)),
               ordered = TRUE)
```

```
d$Features_groups <-
  factor(x = d$Features_groups,
         levels = c("GE","METH","CNV"),
         labels = c("Gene expression","Methylation","Copy number variants"),
         ordered = TRUE)

out$g <- ggplot(data = d,
                mapping= aes(x=Features,
                             y=Loadings,
                             col=Features_groups,
                             pch=Features_groups)) +
          facet_wrap(facets = .~PC,
                     scales = "free_y") +
          scale_color_manual(values=c("#47039FCC",
                                      "#FA9E3BCC",
                                      "#21908C80")) +
          scale_x_continuous(name = "Omic features (x1e4)",
                             breaks=seq(0,6e4,1e4),
                             labels = 0:6) +
          scale_y_continuous(name = "Right eigenvectors (Loadings)") +
          geom_point() +
          theme_minimal() +
          theme(legend.position = 'bottom') +
          labs(col='Omic\nblocks',
               pch='Omic\nblocks')
```



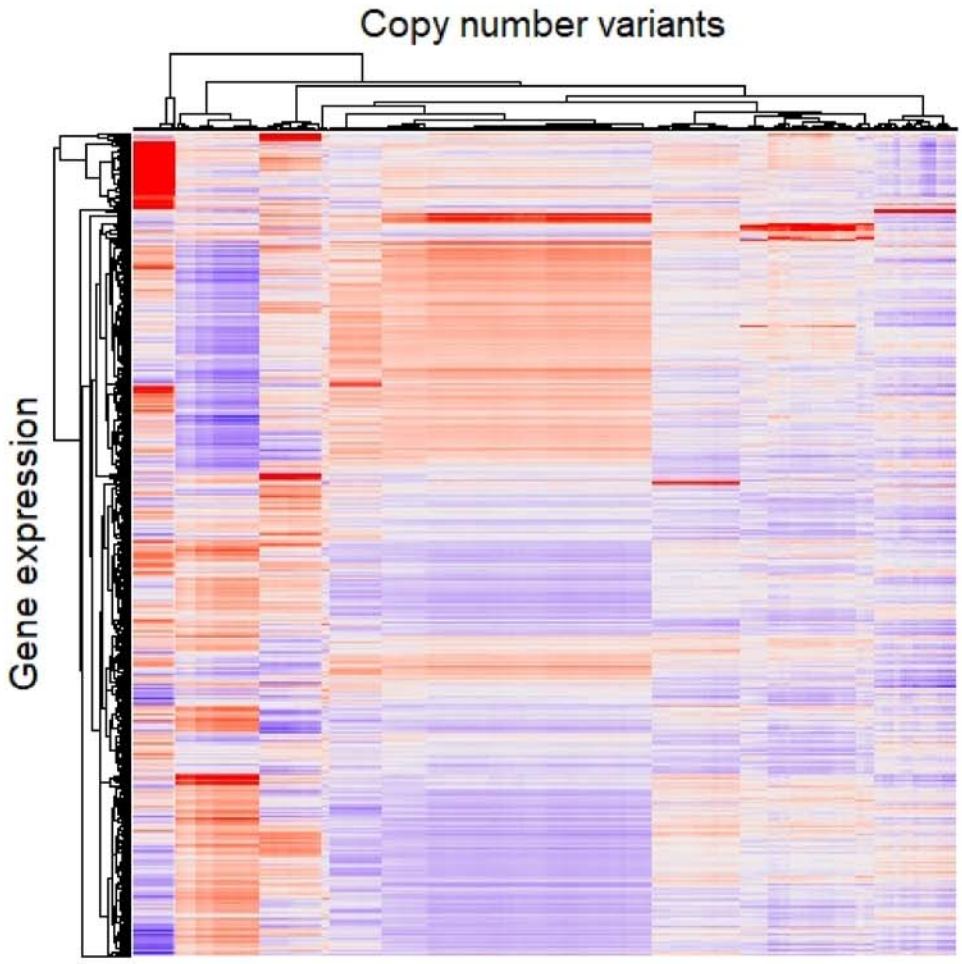
Assessing pan-cancer associations between gene expression and copy number variants.

The code bellow shows how to fit a sparse reduced rank regression model. In this case, we have used elastic net penalty using the *ds* value selected in the above example. Notice that, in this case, the number of latent factors $K.X$ will be used to approximate the **CNV** data so it is easy to compute the inverses needed to obtain **B**. Ideally, the user would want a higher value to assure the approximation is good enough. The value of $K.Y$, on the other hand, will control how many eigenvectors will be used to approximate **B**. In this case, the left eigenvectors represent **CNV**, and the right ones, **GE**. Sparsity in these vectors can then be used to select what combination of genes and CNVs have effects different from zero.

```
set.seed(seed = 347)
out2 <- moss(data.blocks = omic_blocks[-2],
             resp.block = 1,
             method = 'rrr',
             scale. = TRUE,
             norm. = TRUE,
             K.X = 50,
             K.Y = 5,
             dg.grid.left = 3,
             dg.grid.right = 3,
             alpha.right = 0.5,
             alpha.left = 0.5,
             plot = TRUE)

B.sub <- t(x = out$B[which(x = rowMeans(x = out2$sparse$u != 0) > 0),
which(rowMeans(out2$sparse$v != 0) > 0)])

h <- ComplexHeatmap::Heatmap(matrix = B.sub,
                         column_title = "Copy number variants",
                         row_title = "Gene expression",
                         show_column_names = FALSE,
                         show_row_names = FALSE,
                         show_heatmap_legend = FALSE)
```



Session information.

```
sessionInfo()

R version 3.6.2 (2019-12-12)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows >= 8 x64 (build 9200)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.1252  LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] ggpubr_0.2.4    magrittr_1.5    MOSS_0.1.0      ggplot2_3.2.1   bigstatsr_1.1.4
```

```
testthat_2.3.1

loaded via a namespace (and not attached):
 [1] fs_1.3.1             flock_0.7            usethis_1.5.1        devtools_2.2.1
 [5] doParallel_1.0.15    RColorBrewer_1.1-2   rprojroot_1.3-2      tools_3.6.2
 [9] backports_1.1.5      R6_2.4.1             lazyeval_0.2.2       colorspace_1.4-1
[13] GetoptLong_0.1.8     withr_2.1.2          tidyselect_0.2.5     gridExtra_2.3
[17] prettyunits_1.1.1    processx_3.4.1       compiler_3.6.2       bigparallelr_0.2.3
[21] cli_2.0.1            xml2_1.2.2           desc_1.2.0           labeling_0.3
[25] scales_1.1.0         callr_3.4.1          stringr_1.4.0        digest_0.6.23
[29] dbscan_1.1-5         rmarkdown_2.1        htmltools_0.4.0      pkgconfig_2.0.3
[33] sessioninfo_1.1.1    rlang_0.4.3          GlobalOptions_0.1.1  ggthemes_4.2.0
[37] rstudioapi_0.10      shape_1.4.4          farver_2.0.3         dplyr_0.8.3
[41] Matrix_1.2-18        Rcpp_1.0.3           munsell_0.5.0        fansi_0.4.1
[45] viridis_0.5.1        lifecycle_0.1.0      stringi_1.4.3        yaml_2.2.1
[49] ggpmisc_0.3.3        pkgbuild_1.0.6       Rtsne_0.15           plyr_1.8.5
[53] grid_3.6.2           parallel_3.6.2       crayon_1.3.4         lattice_0.20-38
[57] cowplot_1.0.0        circlize_0.4.8       knitr_1.27           ComplexHeatmap_2.2.0
[61] ps_1.3.0             pillar_1.4.3         rjson_0.2.20         ggsignif_0.6.0
[65] reshape2_1.4.3       codetools_0.2-16     pkgload_1.0.2        bigassertr_0.1.2
[69] glue_1.3.1           evaluate_0.14        remotes_2.1.0        splus2R_1.2-2
[73] png_0.1-7            foreach_1.4.7        gtable_0.3.0         purrr_0.3.3
[77] clue_0.3-57          assertthat_0.2.1     xfun_0.12            RSpectra_0.16-0
[81] roxygen2_7.0.2       viridisLite_0.3.0    tibble_2.1.3         iterators_1.0.12
[85] memoise_1.1.0        cluster_2.1.0        ellipsis_0.3.0
```