An introduction to PhosR package

Taiyun Kim [1,2,3]***, Hani Jieun Kim*** [1,2,3]***, Di Xiao*** [2]***, and Pengyi Yang*** [1,2,3]

[1]School of Mathematics and Statistics, The University of Sydney
[2]Computational Systems Biology Group, Children's Medical Research Institute, Faculty of Medicine and Health, The University of Sydney
[3]Charles Perkins Centre, The University of Sydney

27 October 2020

Package

BiocStyle 2.18.0

Contents

1 Introduction

PhosR is a package for the comprenhensive analysis of phosphoproteomic data. There are two major components to PhosR: processing and downstream analysis. PhosR consists of various processing tools for phosphoproteomic data including filtering, imputation, normalisaton and batch correction, enabling integration of multiple phosphoproteomic datasets. Downstream analytical tools consists of site- and protein-centric pathway analysis to evaluate activities of kinases and signalling pathways, large-scale kinase-substrate annotation from dynamic phosphoproteomic profiling, and visualisation and construction of signalomes present in the phosphoproteomic data of interest.

Below is a schematic overview of main componenets of PhosR, categorised into two broad steps of data analytics - processing and downstream analysis.

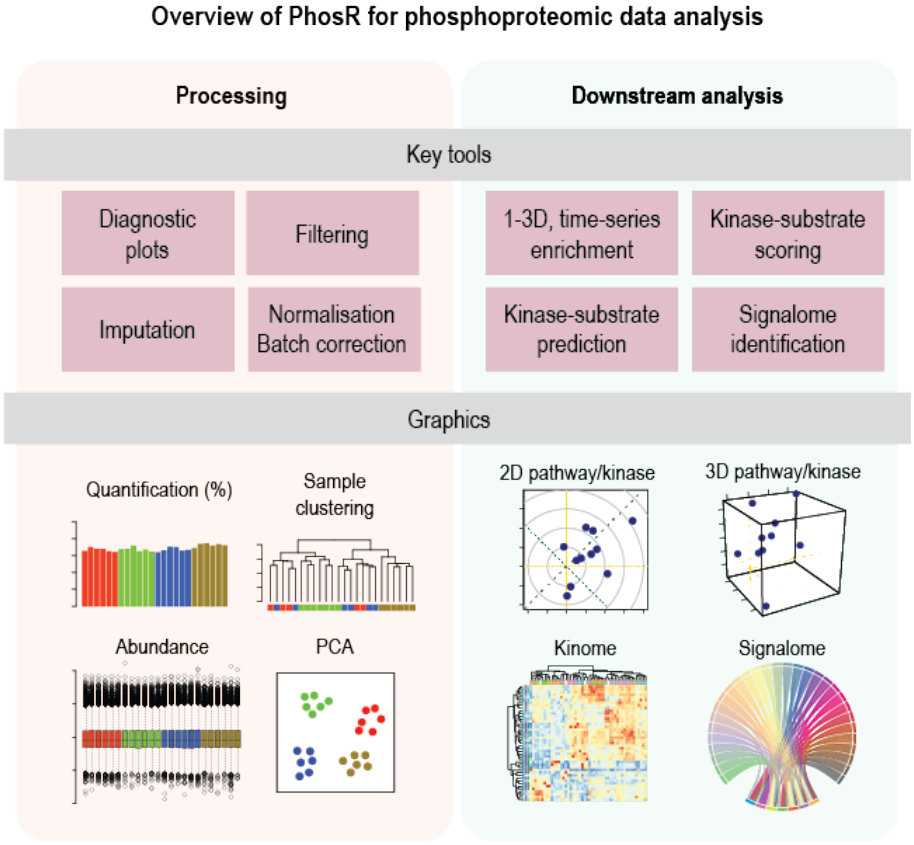


Figure 1: Overview of PhosR methods

The purpose of this vignette is to illustrate some uses of PhosR and explain its key components.

2 Loading packages and data

```
suppressPackageStartupMessages({
    library(PhosR)
```

```
})
```

For demonstration purposes, we provide a rat L6 myotubes phosphoproteome dataset in our package. The data contains ratios of samples treated with AICAR, an analog of adenosine monophosphate that stimulates AMPK activity, insulin (Ins), or in combination (AICAR+Ins) with the basal condition. The full(?) raw data can be found here.

We also provide our novel list of SPSs to be used as a negative control in batch correction and the PhosphoSitePlus annotation for rat, which we will use for XXX analysis below.

For details on how we defined our SPSs can be found in our manuscript, available at XXX

3 Part A. Preprocessing

3.1 Imputation

3.1.1 Introduction

PhosR is a package for the all-rounded analysis of phosphoproteomic data from processing to downstream analysis. This vignette will provide a step-by-step workflow of how PhosR can be used to process and analyse a a panel of phosphoproteomic datasets. As one of the first steps of data processing in phosphoproteomic analysis, we will begin by performing filtering and imputation of phosphoproteomic data with PhosR.

3.1.2 Setting up the data

We assume that you will have the raw data processed using platforms frequently used for mass-spectrometry based proteomics such as MaxQuant. For demonstration purposes, we will take a parts of phosphoproteomic data generated by Humphrey et al. [doi:10.1038/nbt.3327] with accession number PXD001792. The dataset contains the phosphoproteomic quantifications of two mouse liver cell lines (Hepa1.6 and FL38B) that were treated with either PBS (mock) or insulin.

We will take the grouping information from colnames of our matrix.

```
data("phospho.cells.Ins.sample")
grps = gsub("_[0-9]{1}", "", colnames(phospho.cells.Ins))
```

For each cell line, there are two conditions (Control vs Insulin-stimulated) and 6 replicates for each condition.

```
# FL38B
gsub("Intensity.", "", grps)[1:12]
##  [1] "FL83B_Control" "FL83B_Control" "FL83B_Control" "FL83B_Control"
##  [5] "FL83B_Control" "FL83B_Control" "FL83B_Ins"     "FL83B_Ins"
##  [9] "FL83B_Ins"     "FL83B_Ins"     "FL83B_Ins"     "FL83B_Ins"
# Hepa1
gsub("Intensity.", "", grps)[13:24]
##  [1] "Hepa1.6_Control" "Hepa1.6_Control" "Hepa1.6_Control" "Hepa1.6_Control"
```

```
##  [5] "Hepa1.6_Control" "Hepa1.6_Control" "Hepa1.6_Ins"     "Hepa1.6_Ins"
##  [9] "Hepa1.6_Ins"     "Hepa1.6_Ins"     "Hepa1.6_Ins"     "Hepa1.6_Ins"
```

Note that there are in total 24 samples and 5000 phosphosites profiled.

```
dim(phospho.cells.Ins)
## [1] 5000   24
```

3.1.3 Filtering of phosphosites

Next, we will perform some filtering of phosphosites so that only phosphosites with quantifications for at least 50% of the replicates in at least one of the conditions are retained. For this filtering step, we use the `selectGrps` function. The filtering leaves us with 1772 phosphosites.

```
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)
dim(phospho.cells.Ins.filtered)
## [1] 1772   24
```

`selectGrps` gives you the option to relax the threshold for filtering. The filtering threshold can therefore be optimised for each dataset.

```
# In cases where you have fewer replicates ( e.g.,triplicates), you may want to
# select phosphosites quantified in 70% of replicates.
phospho.cells.Ins.filtered1 <- selectGrps(phospho.cells.Ins, grps, 0.7, n=1)
dim(phospho.cells.Ins.filtered1)
## [1] 1330   24
```

3.1.4 Imputation of phosphosites

We can proceed to imputation now that we have filtered for suboptimal phosphosites. To take advantage of data structure and experimental design, PhosR provides users with a lot of flexibility for imputation. There are three functions for imputation: `scImpute`,`tImpute`, and `ptImpute`. Here, we will demonstrate the use of `scImpute` and `ptImpute`.

3.1.5 Site- and condition-specific imputation

The `scImpute` function is used for site- and condition-specific imputation. A predefined thereshold is used to select phosphosites to impute. Phosphosites with missing values equal to or greater than a predefined value will be imputed by sampling from the empirical normal distribution constructed from the quantification values of phosphosites from the same condition.

```
set.seed(123)
phospho.cells.Ins.impute1 <-
    scImpute(phospho.cells.Ins.filtered, 0.5,
        grps)[,colnames(phospho.cells.Ins.filtered)]
```

In the above example, only phosphosites that are quantified in more than 50% of samples from the same condition will be imputed.

3.1.5.1 Paired tail-based imputation

We then perform paired tail-based imputation on the dataset imputed with `scImpute`. Paired tail-based imputation performs imputation of phosphosites that have missing values in *all* replicates in one condition (e.g. in `basal`) but not in another condition (e.g., in `stimulation`). This method of imputation ensures that we do not accidentally filter phosphosites that seemingly have low detection rate, which may be because of true

As for `scImpute`, we can set a predefined threshold to in another condition (e.g. 'stimulation'), the tail-based imputation is applied to impute for the missing values in the first condition.

```
set.seed(123)
phospho.cells.Ins.impute <- phospho.cells.Ins.impute1
phospho.cells.Ins.impute[,1:5] <- ptImpute(phospho.cells.Ins.impute1[,6:10],
                                           phospho.cells.Ins.impute1[,1:5],
                                           percent1 = 0.6, percent2 = 0,
                                           paired = FALSE)
## [1] "idx1: 0"
phospho.cells.Ins.impute[,11:15] <- ptImpute(phospho.cells.Ins.impute1[,16:20],
                                           phospho.cells.Ins.impute1[,11:15],
                                           percent1 = 0.6, percent2 = 0,
                                           paired = FALSE)
## [1] "idx1: 0"
```

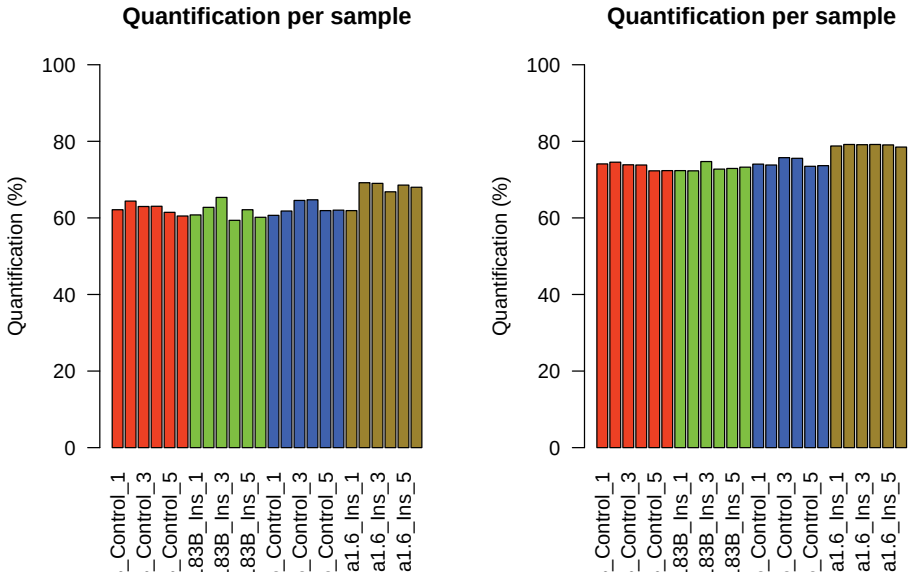
Lastly, we perform normalisation of the filtered and imputed phosphoproteomic data.

```
phospho.cells.Ins.ms <- medianScaling(phospho.cells.Ins.impute, scale = FALSE)
```

3.1.6 Quantification plots

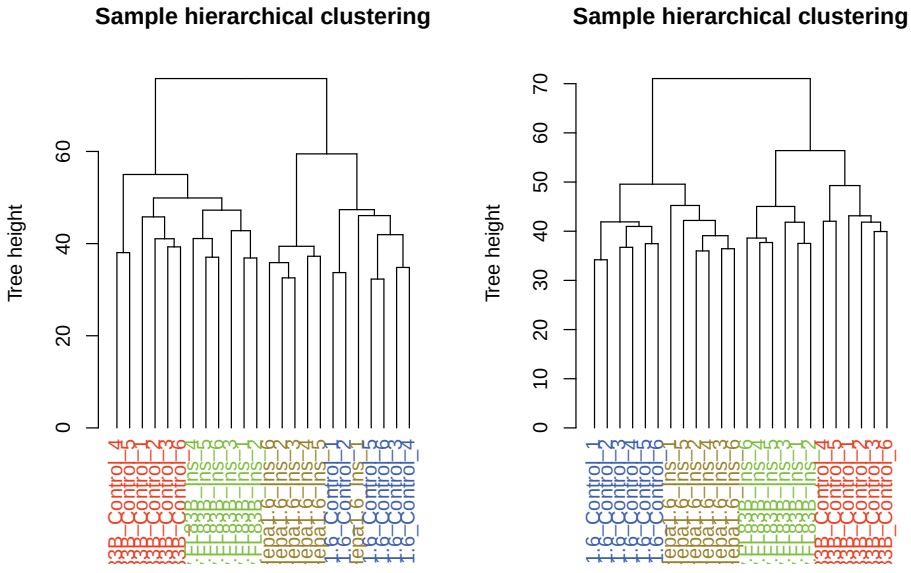
A useful function in `PhosR` is to visualize the percentage of quantified sites before and after filtering and imputation. The main inputs of `plotQC` are the quantification matrix, sample labels (equating the column names of the matrix), an integer indicating the panel to plot, and lastly, a color vector. To visualize the percentage of quantified sites, use the `plotQC` function and set `panel = 1` to visualise barplots of samples.

```
cols <- rep(c("#ED4024", "#7FBF42", "#3F61AD", "#9B822F"), each=6)
par(mfrow=c(1,2))
plotQC(phospho.cells.Ins.filtered, labels=colnames(phospho.cells.Ins.filtered),
        panel = 1, cols = cols)
plotQC(phospho.cells.Ins.ms, labels=colnames(phospho.cells.Ins.ms), panel = 1,
        cols = cols)
```



By setting `panel = 2`, we can visualise the results of unsupervised hierarchical clustering of samples as a dendrogram. The dendrogram demonstrates that imputation has improved the clustering of the samples so that replicates from the same conditions cluster together.

```
par(mfrow=c(1,2))
plotQC(phospho.cells.Ins.filtered, labels=colnames(phospho.cells.Ins.filtered),
       panel = 2, cols = cols)
plotQC(phospho.cells.Ins.ms, labels=colnames(phospho.cells.Ins.ms), panel = 2,
       cols = cols)
```



We can now move onto the next step in the `PhosR` workflow: integration of datasets and batch correction.

3.2 Batch correction

3.2.1 Introduction

A common but largely unaddressed challenge in phosphoproteomic data analysis is to correct for batch effect. Without correcting for batch effect, it is impossible to analyze datasets, derived in batches or from independent labs, in an integrative manner. To perform data integration and batch effect correction, we identified a set of stably phosphorylated sites (SPSs) across a panel of phosphoproteomic datasets and, using these SPSs, implemented a wrapper function of RUV-III from the `ruv` package called `RUVphospho`.

Note that when the input data contains missing values, imputation should be performed before batch correction since RUV-III requires a complete data matrix. The imputed values are removed by default after normalisation but can be retained for downstream analysis if the users wish to use the imputed matrix. This vignette will provide an example of how PhosR can be used for batch correction.

In this example, we will use L6 myotube phosphoproteome dataset (with accession number PXD019127) and the SPSs we identified from a panel of phosphoproteomic datasets (please refer to our preprint for the full list of the datasets used). The `SPSs` will be used as our `negative control` during RUV normalisation.

```
data("phospho_L6_ratio")
data("SPSs")
```

3.2.2 Setting up the data

The L6 myotube data contains phosphoproteomic samples from three treatment conditions each with quadruplicates. Myotube cells were treated with either AICAR or Insulin (Ins), which are both important modulators of the insulin signalling pathway, or both (AICARIns) before phosphoproteomic analysis.

```
colnames(phospho.L6.ratio)[grepl("AICAR_", colnames(phospho.L6.ratio))]
## [1] "AICAR_exp1" "AICAR_exp2" "AICAR_exp3" "AICAR_exp4"
colnames(phospho.L6.ratio)[grepl("^Ins_", colnames(phospho.L6.ratio))]
## [1] "Ins_exp1" "Ins_exp2" "Ins_exp3" "Ins_exp4"
colnames(phospho.L6.ratio)[grepl("AICARIns_", colnames(phospho.L6.ratio))]
## [1] "AICARIns_exp1" "AICARIns_exp2" "AICARIns_exp3" "AICARIns_exp4"
```

Note that we have in total 6654 quantified phosphosites and 12 samples in total.

```
dim(phospho.L6.ratio)
## [1] 6660   12
```

We have already performed the revelant processing steps to generate a dense matrix. Please refer to `imputation` page to perform filtering and imputation of phosphosites in order to generate a matrix without any missing values.

```
sum(is.na(phospho.L6.ratio))
## [1] 0
```

We will clean up the phosphosite labels, which currently contain many unnecessary information for our current analysis (e.g., phosphosite sequence).

```
# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
head(phospho.site.names)
## [1] "Q6AYR1~Tfg~S198~MSAFGLTDDQVSGPPSAPTEDRSGTPDSIAS"
## [2] "D3ZRN2~Med1~T1035~STGGSKSPGSSGRCQTPPGVATPPIPKITIQ"
## [3] "D3ZUD5~Ofd1~S780~SSSPCLDRPSESPAASPTPCPERTPQPSSVP"
## [4] "Q68FR3~Ints12~S127~DVPKKPRLEKPETRSSPITVQTSKDLAMADL"
## [5] "B5DF98~Map3k3~S175~PRSRHLSVSSQNPGRSSPPPGYVPERQQHIA"
## [6] "D3ZPU4~Ercc6l2~S913~RVPKNPICCKLLLGESEDSETEDPVKVNHDD"
```

We will almost remove any duplicate sites.

```
L6.sites = gsub(" ", "", sapply(strsplit(rownames(phospho.L6.ratio), "~"),
                                function(x){paste(toupper(x[2]), x[3], "",
                                                  sep=";")}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
                            colMeans))
head(rownames(phospho.L6.ratio))
## [1] "AAAS;S495;"  "AAGAB;S210;" "AAK1;S18;"   "AAK1;S20;"   "AAK1;S619;"
## [6] "AAK1;S624;"
phospho.site.names = split(phospho.site.names, L6.sites)
```

Lastly, we will take the grouping information from `colnames` of our matrix.

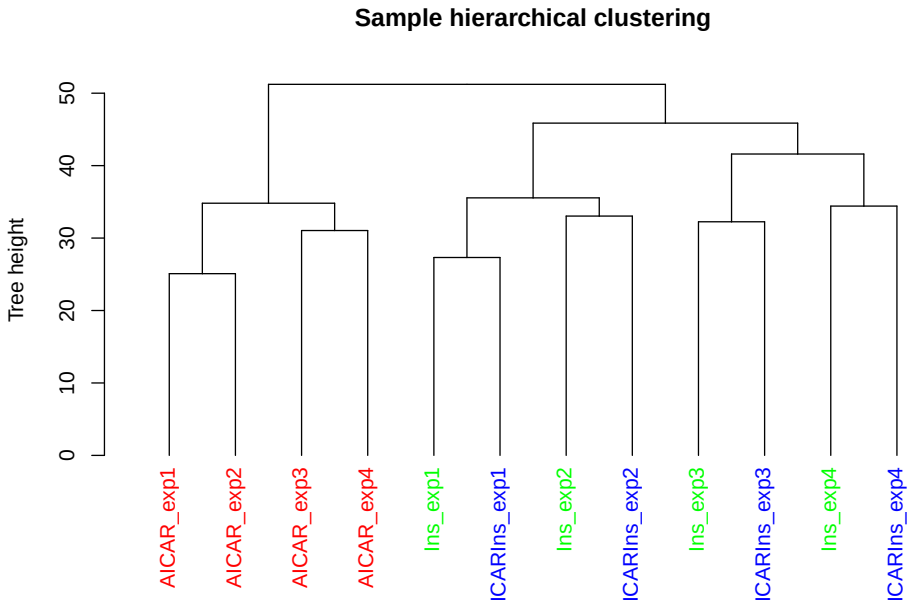
```
# take the grouping information
grps = gsub("_.+", "", colnames(phospho.L6.ratio))
grps
##  [1] "AICAR"    "AICAR"    "AICAR"    "AICAR"    "Ins"      "Ins"
##  [7] "Ins"      "Ins"      "AICARIns" "AICARIns" "AICARIns" "AICARIns"
```

3.2.3 Diagnosing batch effect

There are a number of ways to diagnose batch effect. In `PhosR`, we make use of two visualisation methods to detect batch effect: dendrogram of hierachical clustering and a principal component analysis (PCA) plot. We use the `plotQC` function we introduced in the `imputation` section of the vignette.

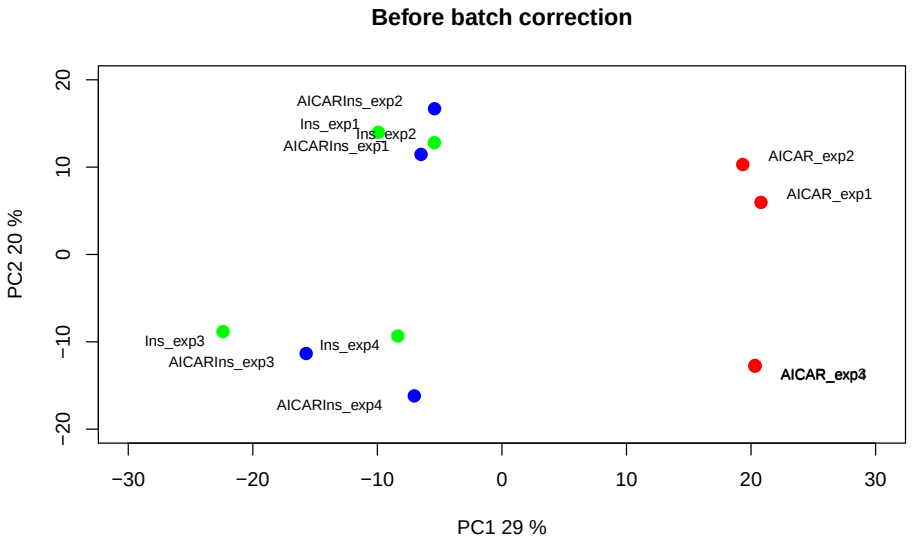
By setting `panel = 2`, we can plot the dendrogram illustrating the results of unsupervised hierarchical clustering of our 12 samples. Clustering results of the samples demonstrate that there is strong batch effect by batch (denoted as `expX`, where X refers to the batch number). This is particularly evident for samples from `Ins` and `AICARIns` treated conditions.

```
cs = rainbow(length(unique(grps)))
colorCodes = sapply(grps, switch, AICAR=cs[1], Ins=cs[2], AICARIns=cs[3])

par(mfrow=c(1,1))
plotQC(phospho.L6.ratio, panel = 2, cols=colorCodes,
       main = "Before batch correction")
```



We can also visualise the samples in PCA space by setting `panel = 4`. The PCA plot demonstrates aggregation of samples by batch rather than treatment groups (each point represents a sample coloured by treatment condition). It has become clearer that even within the `AICAR` treated samples, there is some degree of batch effect as data points are separated between samples from batches 1 and 2 and those from batches 3 and 4.

```
par(mfrow=c(1,1))
plotQC(phospho.L6.ratio, cols=colorCodes, labels = colnames(phospho.L6.ratio),
       panel = 4, ylim=c(-20, 20), xlim=c(-30, 30),
       main = "Before batch correction")
```



3.2.4 Correcting batch effect

We have now diagnosed that our dataset exhibits batch effect that is driven by experiment runs for samples treated with three different conditions. To address this batch effect, we correct for this unwanted variation in the data by utilising our novel SPSs as a negative control for `RUVphospho`.

First, we construct a design matrix by condition.

```
design = model.matrix(~ grps - 1)
design
##    grpsAICAR grpsAICARIns grpsIns
## 1          1            0       0
## 2          1            0       0
## 3          1            0       0
## 4          1            0       0
## 5          0            0       1
## 6          0            0       1
## 7          0            0       1
## 8          0            0       1
## 9          0            1       0
## 10         0            1       0
## 11         0            1       0
## 12         0            1       0
## attr(,"assign")
## [1] 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$grps
## [1] "contr.treatment"
```

We will then use the `RUVphospho` function to normalise the data. Besides the quantification matrix and the design matrix, there are two other important inputs to `RUVphospho`: 1) the `ctl` argument is an integer vector denoting the position of SPSs within the quantification matrix 2) `k` parameter is an integer denoting the expected number of experimental (e.g., treatment) groups within the data

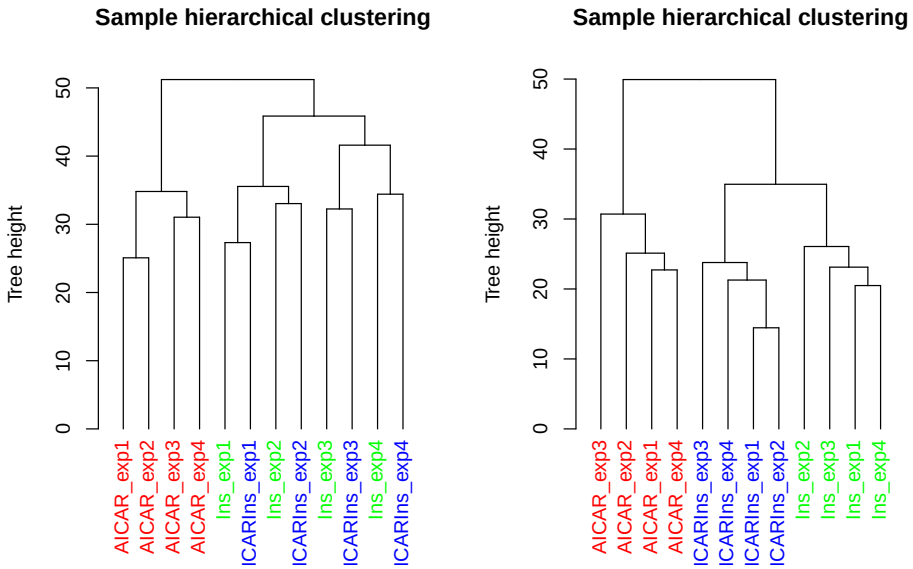
```
# phosphoproteomics data normalisation and batch correction using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
                                  ctl = ctl)
```

3.2.5 Quality control

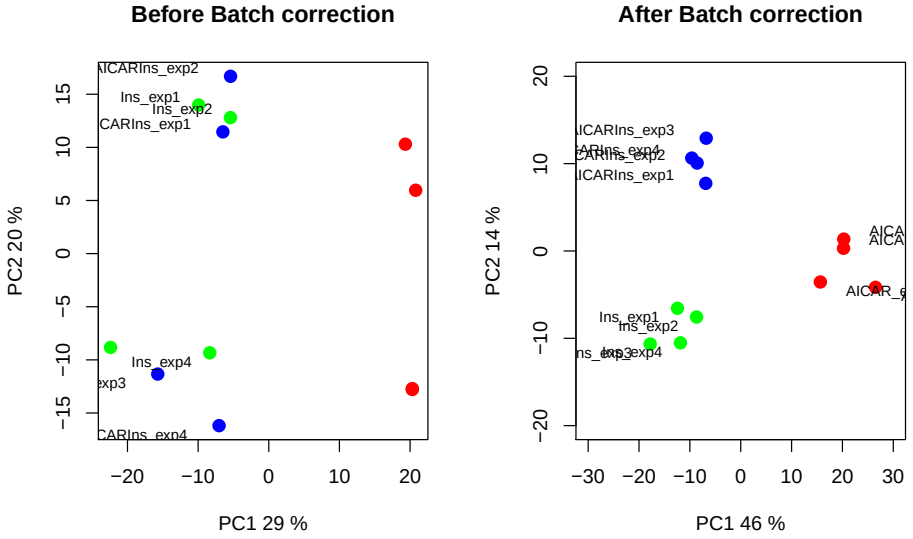
As quality control, we will demonstrate and evaluate our normalisation method with hierarchical clustering and PCA plot using again `plotQC`. Both the hierachical clustering and PCA results demonstrate the normalisation procedure in `PhosR` facilitates effective batch correction.

```
# plot after batch correction
par(mfrow=c(1,2))
plotQC(phospho.L6.ratio, panel = 2, cols=colorCodes)
plotQC(phospho.L6.ratio.RUV, cols=colorCodes,
```

```
        labels = colnames(phospho.L6.ratio), panel=2, ylim=c(-20, 20),
        xlim=c(-30, 30))
```




```
par(mfrow=c(1,2))
plotQC(phospho.L6.ratio, panel = 4, cols=colorCodes,
        labels = colnames(phospho.L6.ratio), main="Before Batch correction")
plotQC(phospho.L6.ratio.RUV, cols=colorCodes,
        labels = colnames(phospho.L6.ratio), panel=4, ylim=c(-20, 20),
        xlim=c(-30, 30), main="After Batch correction")
```



4 Part B. Downstream analysis

4.1 Pathway analysis

4.1.1 Introduction

Most phosphoproteomic studies have adopted a phosphosite-level analysis of the data. To enable phosphoproteomic data analysis on the gene level, `PhosR` implements both site- and gene-centric analyses for detecting changes in kinase activities and signalling pathways through traditional enrichment analyses (over-representation or rank-based gene set test, together referred to as '1-dimensional enrichment analysis') as well as 2- and 3-dimensional analyses.

This vignette will perform gene-centric pathway enrichment analyses on the normalised myotube phosphoproteomic dataset using both over-representation and rank–based gene set tests and also provide an example of how `directPA` can be used to test which kinases are activated upon different stimulations in myotubes using 2-dimensional analyses. ([Pengyi Yang et al. 2014]) https://academic.oup.com/bioinformatics/article/30/6/808/286146.

4.1.2 Loading packages and data

First, we will load the PhosR package with few other packages will use for the demonstration purpose.

We will use RUV normalised L6 phosphopreteome data for demonstration of gene-centric pathway analysis. It contains phosphoproteome under three different treatment conditions and a basal condition, and three conditions are (1) AMPK agonist AICAR, (2) insulin (Ins), (3) in combination (AICAR+Ins).

```
suppressPackageStartupMessages({
    library(calibrate)
    library(limma)
    library(directPA)
})
```

```
data("PhosphoSitePlus")
```

We will use `phospho.L6.ratio.RUV` matrix from Section 1.2 Batch correction.

4.1.3 1-dimensional enrichment analysis

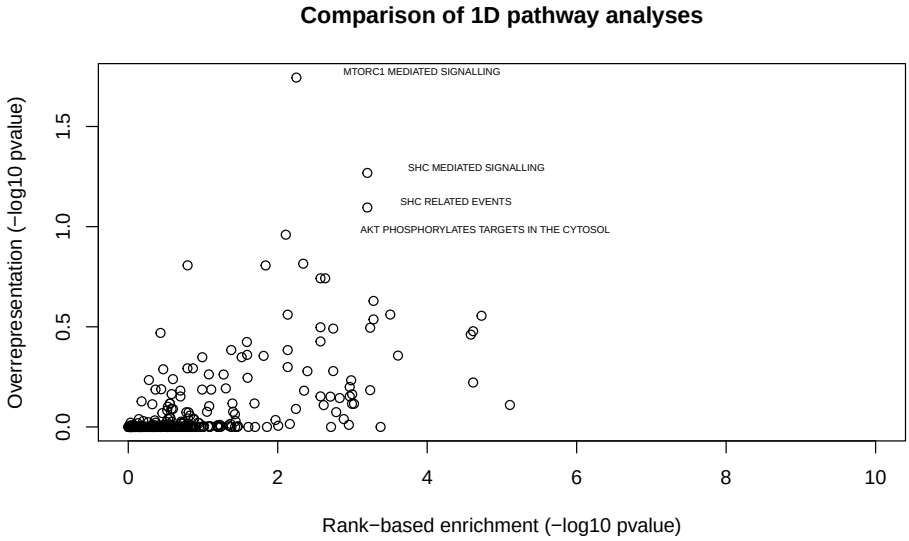
To enable enrichment analyses on both gene and phosphosite levels, PhosR implements a simple method called `phosCollapse` which reduces phosphosite level of information to the proteins for performing downstream gene-centric analyses. We will utilise two functions, `pathwayOverrepresent` and `pathwayRankBasedEnrichment`, to demonstrate 1-dimensional (over-representation and rank-based gene set test) gene-centric pathway enrichment analysis respectively.

```
# divides the phospho.L6.ratio data into groups by phosphosites
L6.sites <- gsub(" ", "", gsub("~[STY]", "~",
                               sapply(strsplit(rownames(phospho.L6.ratio.RUV),
```

```
                                                    "~"),
                                        function(x){paste(toupper(x[2]), x[3],
                                                          sep="~")})))
phospho.L6.ratio.sites <- t(sapply(split(data.frame(phospho.L6.ratio.RUV),
                                         L6.sites), colMeans))

# fit linear model for each phosphosite
f <- gsub("_exp\\d", "", colnames(phospho.L6.ratio.RUV))
X <- model.matrix(~ f - 1)
fit <- lmFit(phospho.L6.ratio.RUV, X)

# extract top-ranked phosphosites for each condition compared to basal
table.AICAR <- topTable(eBayes(fit), number=Inf, coef = 1)
table.Ins <- topTable(eBayes(fit), number=Inf, coef = 3)
table.AICARIns <- topTable(eBayes(fit), number=Inf, coef = 2)

DE1.RUV <- c(sum(table.AICAR[,"adj.P.Val"] < 0.05),
             sum(table.Ins[,"adj.P.Val"] < 0.05),
             sum(table.AICARIns[,"adj.P.Val"] < 0.05))

# extract top-ranked phosphosites for each group comparison
contrast.matrix1 <- makeContrasts(fAICARIns-fIns, levels=X)
contrast.matrix2 <- makeContrasts(fAICARIns-fAICAR, levels=X)
fit1 <- contrasts.fit(fit, contrast.matrix1)
fit2 <- contrasts.fit(fit, contrast.matrix2)
table.AICARInsVSIns <- topTable(eBayes(fit1), number=Inf)
table.AICARInsVSAICAR <- topTable(eBayes(fit2), number=Inf)

DE2.RUV <- c(sum(table.AICARInsVSIns[,"adj.P.Val"] < 0.05),
             sum(table.AICARInsVSAICAR[,"adj.P.Val"] < 0.05))

o <- rownames(table.AICARInsVSIns)
Tc <- cbind(table.Ins[o,"logFC"], table.AICAR[o,"logFC"],
             table.AICARIns[o,"logFC"])
rownames(Tc) = gsub("(.*)(;[A-Z])([0-9]+)(;)", "\\1;\\3;", o)
colnames(Tc) <- c("Ins", "AICAR", "AICAR+Ins")

# summary phosphosite-level information to proteins for performing downstream
# gene-centric analyses.
Tc.gene <- phosCollapse(Tc, id=gsub(";.+", "", rownames(Tc)),
                        stat=apply(abs(Tc), 1, max), by = "max")
geneSet <- names(sort(Tc.gene[,1],
                        decreasing = TRUE))[1:round(nrow(Tc.gene) * 0.1)]

# 1D gene-centric pathway analysis
path1 <- pathwayOverrepresent(geneSet, annotation=Pathways.reactome,
                                universe = rownames(Tc.gene), alter = "greater")
path2 <- pathwayRankBasedEnrichment(Tc.gene[,1],
                                    annotation=Pathways.reactome,
                                    alter = "greater")
```

Next, we will compare enrichment of pathways (in negative log10 p-values) between the two 1-dimensional pathway enrichment analysis. On the scatter plot, the x-axis and y-axis refer to the p-values derived from the rank-based gene set test and over-representation test, respectively. We find several expected pathways, while these highly enriched pathways are largely in agreement between the two types of enrichment analyses.

```
lp1 <- -log10(as.numeric(path2[names(Pathways.reactome),1]))
lp2 <- -log10(as.numeric(path1[names(Pathways.reactome),1]))
plot(lp1, lp2, ylab="Overrepresentation (-log10 pvalue)",
        xlab="Rank-based enrichment (-log10 pvalue)",
        main="Comparison of 1D pathway analyses", xlim = c(0, 10))

# select highly enriched pathways
sel <- which(lp1 > 1.5 & lp2 > 0.9)
textxy(lp1[sel], lp2[sel], gsub("_", " ", gsub("REACTOME_", "",
                                                names(Pathways.reactome)))[sel])
```



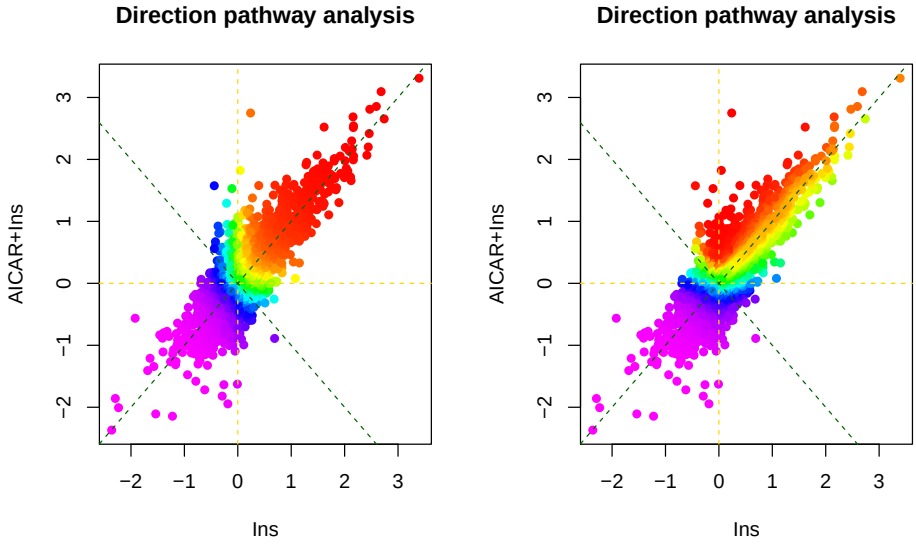
4.1.4 2- and 3-dimensional signalling pathway analysis

One key aspect in studying signalling pathways is to identify key kinases that are involved in signalling cascades. To identify these kinases, we make use of kinase-substrate annotation databases such as `PhosphoSitePlus` and `Phospho.ELM`. These databases are included in the `PhosR` and `directPA` packages already. To access them, simply load the package and access the data by `data("PhosphoSitePlus")` and `data("PhosphoELM")`.

The 2- and 3-dimensional analyses enable the investigation of kinases regulated by different combinations of treatments. We will introduce more advanced methods implemented in the R package `directPA` for performing "2 and 3-dimentional" direction site-centric kinase activity analyses.

```
# 2D direction site-centric kinase activity analyses
par(mfrow=c(1,2))
dpa1 <- directPA(Tc[,c(1,3)], direction=0,
                 annotation=lapply(PhosphoSite.rat,
```

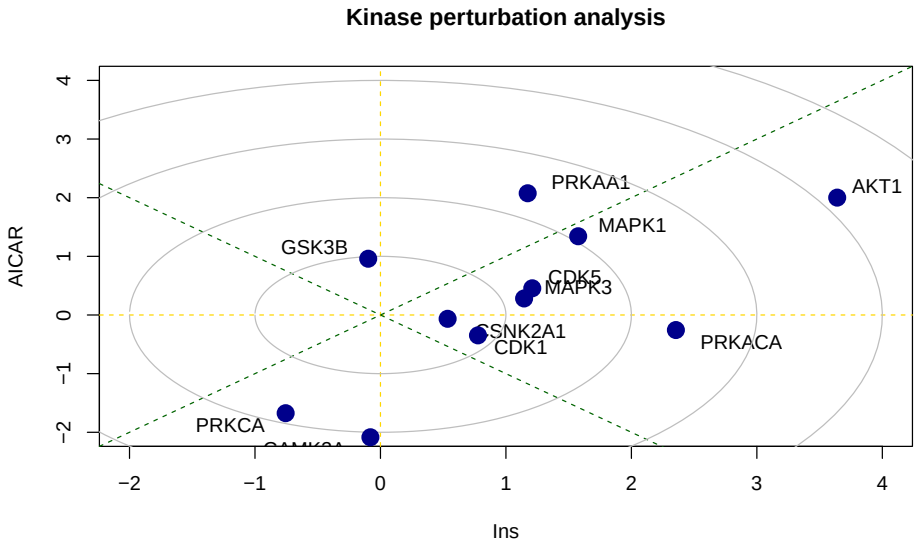
```
                                    function(x){gsub(";[STY]", ";", x)}),
                  main="Direction pathway analysis")
dpa2 <- directPA(Tc[,c(1,3)], direction=pi*7/4,
                  annotation=lapply(PhosphoSite.rat,
                                    function(x){gsub(";[STY]", ";", x)}),
                  main="Direction pathway analysis")
```



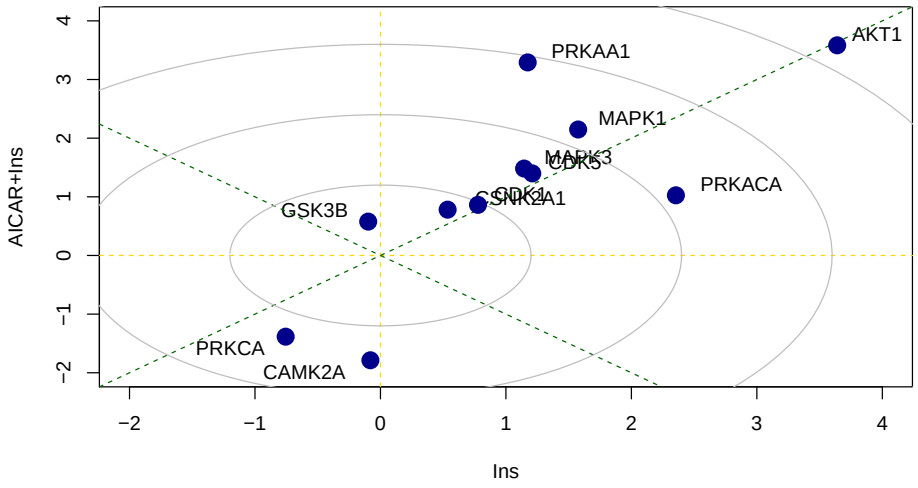

```
# top activated kinases
dpa1$pathways[1:5,]
##         pvalue       size
## AKT1   6.207001e-09 9
## MAPK1  0.00057404   9
## PRKACA 0.0006825021 25
## PRKAA1 0.000965093  6
## MAPK3  0.006670176  10
dpa2$pathways[1:5,]
##          pvalue     size
## PRKAA1  0.00463462 6
## AKT1    0.02942273 9
## CSNK2A1 0.2193148  12
## CDK5    0.2607434  5
## MAPK1   0.2767886  9
```

There is also a function called `perturbPlot2d` implemented in `kinasePA` for testing and visualising activity of all kinases on all possible directions. Below are the demonstration from using this function.

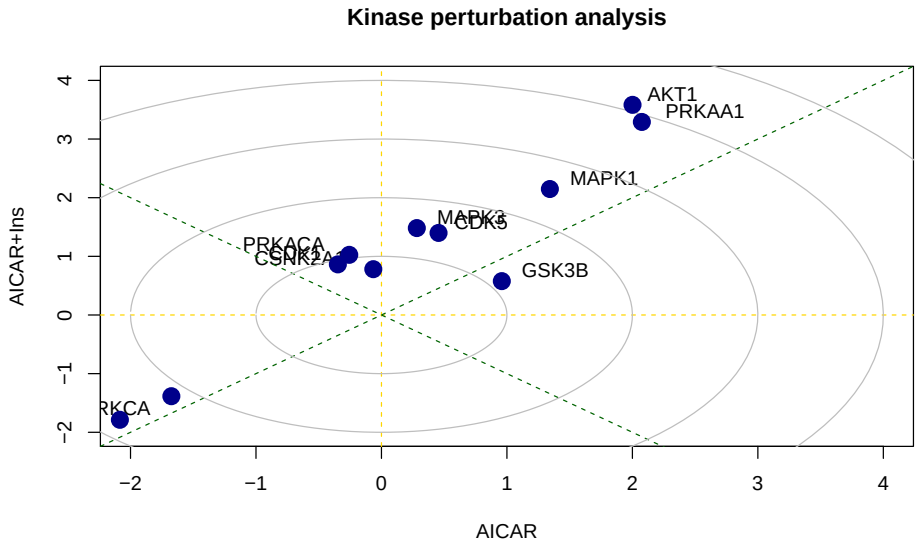
```
z1 <- perturbPlot2d(Tc=Tc[,c(1,2)],
                    annotation=lapply(PhosphoSite.rat,
                                      function(x){gsub(";[STY]", ";", x)}),
                    cex=1, xlim=c(-2, 4), ylim=c(-2, 4),
                    main="Kinase perturbation analysis")
```




```
z2 <- perturbPlot2d(Tc=Tc[,c(1,3)],
                    annotation=lapply(PhosphoSite.rat,
                                      function(x){gsub(";[STY]", ";", x)}),
                    cex=1, xlim=c(-2, 4), ylim=c(-2, 4),
                    main="Kinase perturbation analysis")
```




```
z3 <- perturbPlot2d(Tc=Tc[,c(2,3)],
                    annotation=lapply(PhosphoSite.rat,
                                      function(x){gsub(";[STY]", ";", x)}),
                    cex=1, xlim=c(-2, 4), ylim=c(-2, 4),
                    main="Kinase perturbation analysis")
```



4.2 Site- and gene- centric analysis

4.2.1 Introduction

While 1, 2, and 3D pathway analyses are useful for data generated from experiments with different treatment/conditions, analysis designed for time-course data may be better suited to analysis experiments that profile multiple time points.

Here, we will apply `ClueR` which is an R package specifically designed for time-course proteomic and phosphoproteomic data analysis ([Pengyi Yang et al. 2015])(https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004403).

4.3 Loading packages and data

We will load few other packages we will use for the demonstration purpose.

```
suppressPackageStartupMessages({
    library(parallel)
    library(ggplot2)
    library(ClueR)
})
```

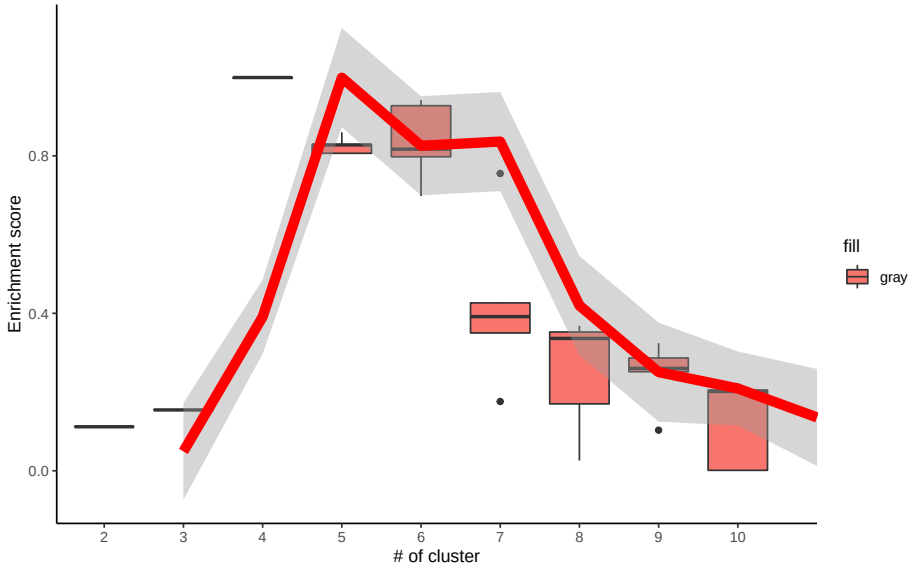
We will load a dataset integrated from two time-course datasets of early and intermediate insulin signalling in mouse liver upon insulin stimulation to demonstrate the time-course phosphoproteomic data analyses.

```
data("phospho_liverInsTC_RUV_sample")
```

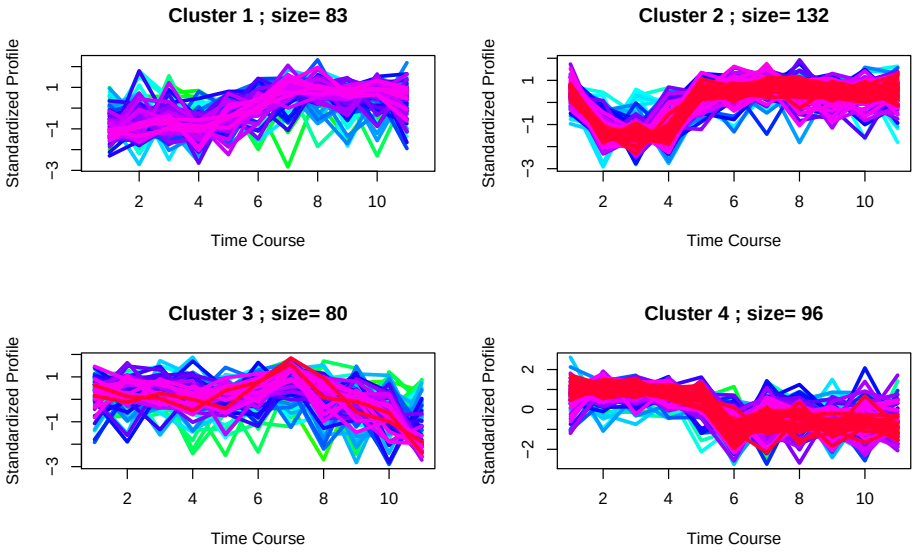
4.4 Gene-centric analyses of the liver phosphoproteome data

Let us start with gene-centric analysis. Such analysis can be directly applied to proteomics data. It can also be applied to phosphoproteomic data by using the `phosCollapse` function to summarise phosphosite information to proteins.

```
rownames(phospho.liver.Ins.TC.ratio.RUV) <-
    sapply(strsplit(rownames(phospho.liver.Ins.TC.ratio.RUV), "~"),
        function(x)paste(x[1], x[2], "", sep=";"))

# take grouping information
grps <- sapply(strsplit(colnames(phospho.liver.Ins.TC.ratio.RUV), "_"),
            function(x)x[3])

# select differentially phosphorylated sites
sites.p <- matANOVA(phospho.liver.Ins.TC.ratio.RUV, grps)
phospho.LiverInsTC <- meanAbundance(phospho.liver.Ins.TC.ratio.RUV, grps)
sel <- which((sites.p < 0.05) & (rowSums(abs(phospho.LiverInsTC) > 1) != 0))
phospho.LiverInsTC.sel <- phospho.LiverInsTC[sel,]

# summarise phosphosites information into gene level
phospho.liverInsTC.gene <-
    phosCollapse(phospho.LiverInsTC.sel,
            gsub(";.+", "", rownames(phospho.LiverInsTC.sel)),
                stat = apply(abs(phospho.LiverInsTC.sel), 1, max), by = "max")

# perform ClueR to identify optimal number of clusters
RNGkind("L'Ecuyer-CMRG")
set.seed(123)
c1 <- runClue(phospho.liverInsTC.gene, annotation=Pathways.reactome,
            kRange = 2:10, rep = 5, effectiveSize = c(5, 100),
            pvalueCutoff = 0.05, alpha = 0.5)

# Visualise the evaluation results
data <- data.frame(Success=as.numeric(c1$evlMat), Freq=rep(2:10, each=5))
myplot <- ggplot(data, aes(x=Freq, y=Success)) +
    geom_boxplot(aes(x = factor(Freq), fill="gray")) +
    stat_smooth(method="loess", colour="red", size=3, span = 0.5) +
    xlab("# of cluster") +
    ylab("Enrichment score") +
    theme_classic()
myplot
## `geom_smooth()` using formula 'y ~ x'
```




```
set.seed(123)
best <- clustOptimal(c1, rep=5, mfrow=c(2, 2), visualize = TRUE)
```

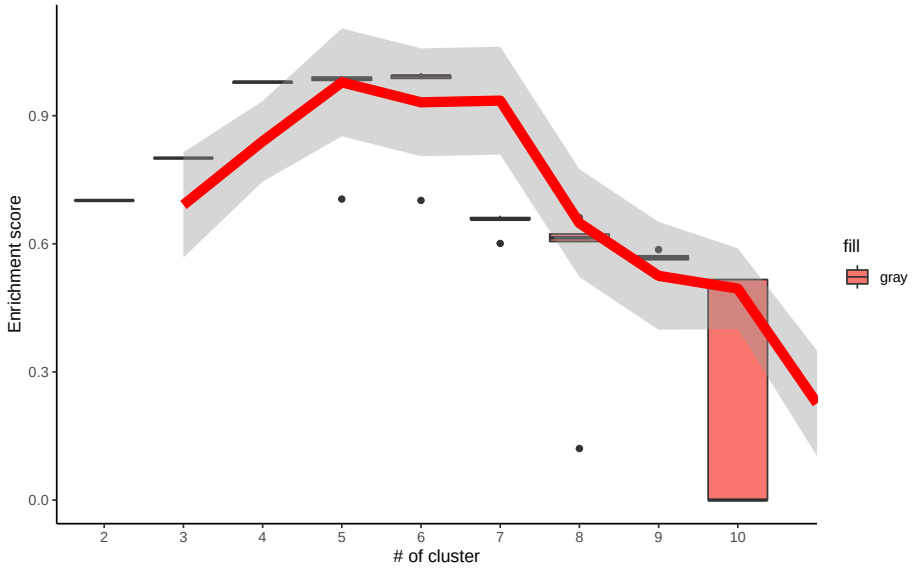



```
# Finding enriched pathways from each cluster
# ps <- sapply(best$enrichList, function(x){
#   l <- ifelse(nrow(x) < 3, nrow(x), 3)
#   x[1:l,2]
# })
# par(mfrow = c(1,1))
# barplot(-log10(as.numeric(unlist(ps))))
```

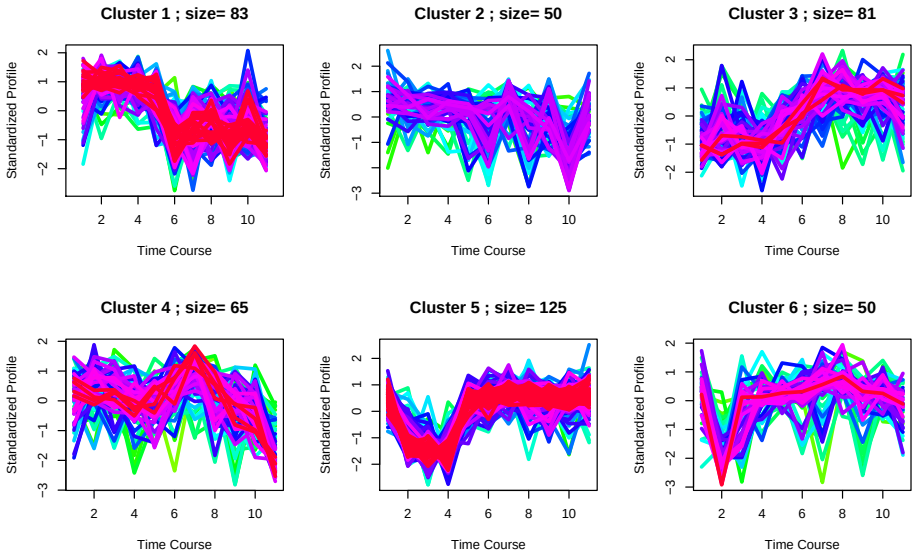
4.5 Site-centric analyses of the liver phosphoproteome data

Phosphosite-centric analyses will perform using kinase-substrate annotation information from PhosphoSitePlus.

```
RNGkind("L'Ecuyer-CMRG")
set.seed(1)
PhosphoSite.mouse2 = mapply(function(kinase) {
    gsub("(.*)(;[A-Z])([0-9]+;)", "\\1;\\3", kinase)
}, PhosphoSite.mouse)

# perform ClueR to identify optimal number of clusters
c3 <- runClue(phospho.LiverInsTC.sel, annotation=PhosphoSite.mouse2,
            kRange = 2:10, rep = 5, effectiveSize = c(5, 100),
            pvalueCutoff = 0.05, alpha = 0.5)

# Visualise the evaluation results
data <- data.frame(Success=as.numeric(c3$evlMat), Freq=rep(2:10, each=5))
myplot <- ggplot(data, aes(x=Freq, y=Success)) +
    geom_boxplot(aes(x = factor(Freq), fill="gray")) +
    stat_smooth(method="loess", colour="red", size=3, span = 0.5) +
    xlab("# of cluster") +
    ylab("Enrichment score") +
    theme_classic()
myplot
## `geom_smooth()` using formula 'y ~ x'
```




```
set.seed(1)
best <- clustOptimal(c3, rep=10, mfrow=c(2, 3), visualize = TRUE)
```




```
# Finding enriched pathways from each cluster
best$enrichList
## $`cluster 1`
##      kinase   pvalue                 size
## [1,] "PRKACA" "0.000184676866298047" "5"
##      substrates
## [1,] "NR1H3;196;|MARCKS;163;|PRKACA;339;|ITPR1;1755;|SIK3;493;"
##
## $`cluster 3`
##      kinase         pvalue                 size
## [1,] "Humphrey.Akt" "0.000162969329853963" "5"
## [2,] "Yang.Akt"     "0.000165386907010959" "6"
##      substrates
## [1,] "TSC2;939;|PFKFB2;486;|FOXO3;252;|FOXO1;316;|GSK3A;21;"
## [2,] "AKT1S1;247;|TSC2;939;|PFKFB2;486;|FOXO3;252;|FOXO1;316;|GSK3A;21;"
```

4.6 Signalomes

4.6.1 Introduction

A key component of the `PhosR` package is to construct signalomes. The signalome construction is composed of two main steps: 1) kinase-substrate relationsip scoring and 2) signalome construction. This involves a sequential workflow where the outputs of the first step are used as inputs of the latter step.

In brief, our kinase-substrate relationship scoring method (`kinaseSubstrateScore` and `kinaseSubstratePred`) prioritises potential kinases that could be responsible for the phosphorylation change of phosphosite on the basis of kinase recognition motif and phosphoproteomic dynamics. Using the kinase-substrate relationships derived from the scoring methods, we reconstruct signalome networks present in the data (`Signalomes`) wherin we highlight kinase regulation of discrete modules.

4.7 Loading packages and data

First, we will load few other packages that we will be using in this section of the vignette.

```
suppressPackageStartupMessages({
    library(dplyr)
    library(ggplot2)
    library(GGally)
    library(ggpubr)
    library(calibrate)
})
```

We will also be needing data containing kinase-substrate annotations from `PhosphoSitePlus`, kinase recognition motifs from `kinase motifs`, and annotations of kinase families from `kinase family`.

```
data("KinaseMotifs")
data("KinaseFamily")
```

4.8 Setting up the data

We will use `phospho.L6.ratio.RUV` matrix from Section 1.2 Batch correction, and we will call it `phosphoL6` from this point for simplicity.

```
phosphoL6 = phospho.L6.ratio.RUV
```

4.9 Generation of kinase-substrate relationship scores

Next, we will filtered for dynamically regulated phosphosites and then standardise the filtered matrix.

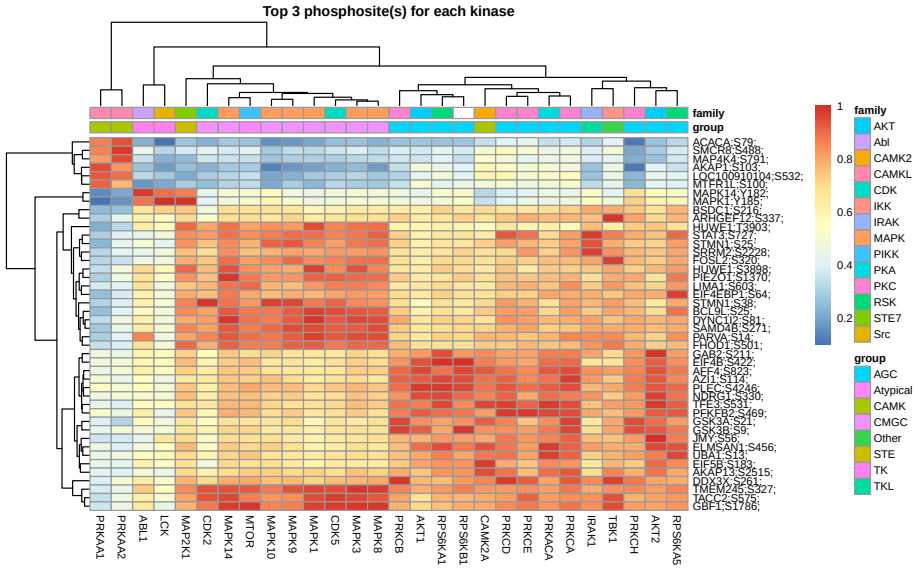
```
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = gsub("_.+", "",
                                                       colnames(phosphoL6)))
aov <- matANOVA(mat=phosphoL6, grps=gsub("_.+", "", colnames(phosphoL6)))
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
                           (rowSums(phosphoL6.mean > 0.5) > 0), ,drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
rownames(L6.phos.std) <-
    sapply(strsplit(rownames(L6.phos.std), "~"),
        function(x){gsub(" ", "", paste(toupper(x[2]), x[3], "", sep=";"))})
```

We next extract the kinase recognition motifs from each of the phosphosites.

```
L6.phos.seq <- sapply(strsplit(rownames(phosphoL6.reg), "~"), function(x)x[4])
```

Now that we have all the inputs for `kinaseSubstrateScore` and `kinaseSubstratePred` ready, we can proceed to the generation of kinase-substrate relationship scores.

```
L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
                                    L6.phos.seq, numMotif = 5, numSub = 1)
## [1] "Number of kinases passed motif size filtering: 114"
## [1] "Number of kinases passed profile size filtering: 44"
## [1] "Scoring phosphosites against kinase motifs:"
## 1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.33.34.35.36.37.38.3
## Scoring phosphosites against kinase-substrate profiles:[1] "done."
## Generating combined scores for phosphosites
## by motifs and phospho profiles:[1] "done."
```

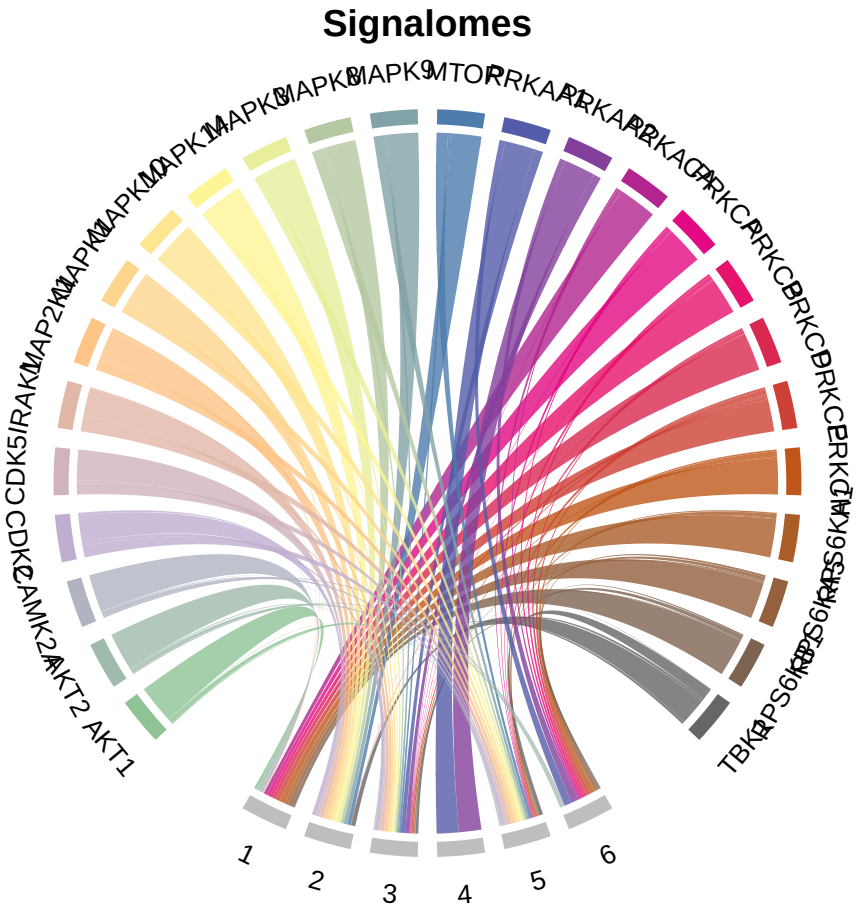



```
set.seed(1)
L6.predMat <- kinaseSubstratePred(L6.matrices, top=30)
## [1] "Predicting kinases for phosphosites:"
## 1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.26.[1] "done"
```

4.10 Signalome construction

The signalome construction uses the outputs of `kinaseSubstrateScore` and `kinaseSubstratePred` functions for the generation of a visualisation of the kinase regulation of discrete regulatory protein modules present in our phosphoproteomic data.

```
kinaseOI = c("PRKAA1", "AKT1")

Signalomes_results <- Signalomes(KSR=L6.matrices,
                                 predMatrix=L6.predMat,
                                 exprsMat=L6.phos.std,
                                 KOI=kinaseOI)
## calculating optimal number of clusters...optimal number of clusters = 3
```

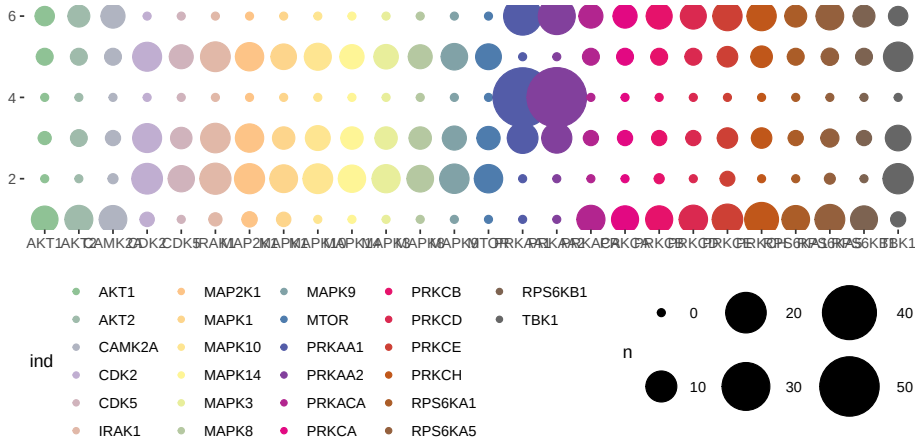


4.11 Generate signalome map

We can also visualise the relative contribution of each kinase towards the regulation of protein modules by plotting a balloon plot. In the balloon plot, the size of the balloons denote the percentage magnitude of kinase regulation in each module.

```
my_color_palette <-
    grDevices::colorRampPalette(RColorBrewer::brewer.pal(8, "Accent"))
kinase_all_color <- my_color_palette(ncol(L6.matrices$combinedScoreMatrix))
names(kinase_all_color) <- colnames(L6.matrices$combinedScoreMatrix)
kinase_signalome_color <- kinase_all_color[colnames(L6.predMat)]

dftoPlot_signalome <- stack(Signalomes_results$kinaseSubstrates)
modules <- Signalomes_results$proteinModule
names(modules) <-
    sapply(strsplit(as.character(names(Signalomes_results$proteinModules)),
                    ";"), "[[", 1)
dftoPlot_signalome$cluster <- modules[dftoPlot_signalome$values]

dftoPlot_balloon_bycluster <- dftoPlot_signalome
dftoPlot_balloon_bycluster <- na.omit(dftoPlot_balloon_bycluster) %>%
```

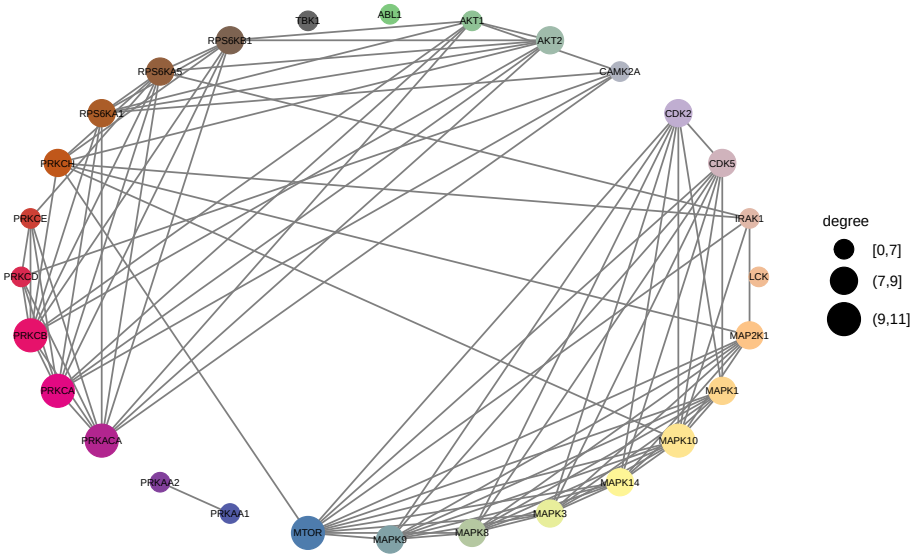
```
        dplyr::count(cluster, ind)
    dftoPlot_balloon_bycluster$ind <- as.factor(dftoPlot_balloon_bycluster$ind)
    dftoPlot_balloon_bycluster$cluster <-
        as.factor(dftoPlot_balloon_bycluster$cluster)
    dftoPlot_balloon_bycluster <-
        tidyr::spread(dftoPlot_balloon_bycluster, ind, n)[,-1]
    dftoPlot_balloon_bycluster[is.na(dftoPlot_balloon_bycluster)] <- 0

    dftoPlot_balloon_bycluster <-
        do.call(rbind, lapply(1:nrow(dftoPlot_balloon_bycluster), function(x) {
            res <- sapply(dftoPlot_balloon_bycluster[x,], function(y)
                y/sum(dftoPlot_balloon_bycluster[x,])*100)
    }))

    dftoPlot_balloon_bycluster <-
        reshape2::melt(as.matrix(dftoPlot_balloon_bycluster))
    colnames(dftoPlot_balloon_bycluster) <- c("cluster", "ind", "n")

    ggplot(dftoPlot_balloon_bycluster, aes(x = ind, y = cluster)) +
        geom_point(aes(col=ind, size=n)) +
        scale_color_manual(values=kinase_signalome_color) +
        scale_size_continuous(range = c(2, 17)) +
        theme_classic() +
        theme(
            aspect.ratio=0.25,
            legend.position = "bottom",
            axis.line = element_blank(),
            axis.title = element_blank(),
            panel.grid.major.x = element_blank(),
            panel.grid.minor.x = element_blank())
```



4.11.1 Generate signalome network

Finally, we can also plot the signalome network that illustrates the connectivity between kinase signalome networks.

```
threskinaseNetwork = 0.9
signalomeKinase <- colnames(L6.predMat)
kinase_cor <- stats::cor(L6.matrices$combinedScoreMatrix)

cor_kinase_mat <- kinase_cor
diag(cor_kinase_mat) <- 0
kinase_network <- lapply(1:ncol(cor_kinase_mat), function(x)
    names(which(cor_kinase_mat[,x] > threskinaseNetwork)))
names(kinase_network) <- colnames(cor_kinase_mat)

cor_kinase_mat <- apply(cor_kinase_mat, 2, function(x) x  > threskinaseNetwork)
cor_kinase_mat[cor_kinase_mat == FALSE] <- 0
cor_kinase_mat[cor_kinase_mat == TRUE] <- 1

library(network)
## network: Classes for Relational Data
## Version 1.16.1 created on 2020-10-06.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##                     Mark S. Handcock, University of California -- Los Angeles
##                     David R. Hunter, Penn State University
##                     Martina Morris, University of Washington
##                     Skye Bender-deMoll, University of Washington
##  For citation information, type citation("network").
##  Type help("network-package") to get started.
links <- reshape2::melt(cor_kinase_mat)
links <- links[links$value == 1,]
res <- sapply(1:length(links$Var1), function(x) {
    kinase_cor[rownames(kinase_cor) == links$Var1[x],
        colnames(kinase_cor) == links$Var2[x]]
})
links$cor <- res
colnames(links) <- c("source", "target", "binary", "cor")

network <- network::network(cor_kinase_mat, directed=FALSE)
GGally::ggnet2(network,
    node.size=10,
    node.color=kinase_all_color,
    edge.size = 0.5,
    size = "degree",
    size.cut=3,
    label=colnames(cor_kinase_mat),
    label.size=2,
    mode="circle",
    label.color="black")
```



5 Session Info

```
sessionInfo()
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.5 LTS
##
## Matrix products: default
## BLAS:   /home/biocbuild/bbs-3.12-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.12-bioc/R/lib/libRlapack.so
##
## Random number generation:
##  RNG:     L'Ecuyer-CMRG
##  Normal:  Inversion
##  Sample:  Rejection
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] network_1.16.1   ggpubr_0.4.0     GGally_2.0.0     dplyr_1.0.2
```

```
##  [5] ClueR_1.4        e1071_1.7-4      ggplot2_3.3.2    directPA_1.4
##  [9] limma_3.46.0     calibrate_1.7.7  MASS_7.3-53      PhosR_1.0.0
## [13] BiocStyle_2.18.0
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-150            webshot_0.5.2           RColorBrewer_1.1-2
##  [4] tools_4.0.3             backports_1.1.10        R6_2.4.1
##  [7] BiocGenerics_0.36.0     mgcv_1.8-33             colorspace_1.4-1
## [10] manipulateWidget_0.10.1 withr_2.3.0             tidyselect_1.1.0
## [13] gridExtra_2.3           curl_4.3                compiler_4.0.3
## [16] preprocessCore_1.52.0   Biobase_2.50.0          labeling_0.4.2
## [19] bookdown_0.21           scales_1.1.1            stringr_1.4.0
## [22] digest_0.6.27           foreign_0.8-80          rmarkdown_2.5
## [25] rio_0.5.16              pkgconfig_2.0.3         htmltools_0.5.0
## [28] fastmap_1.0.1           ruv_0.9.7.1             readxl_1.3.1
## [31] htmlwidgets_1.5.2       rlang_0.4.8             GlobalOptions_0.1.2
## [34] shiny_1.5.0             shape_1.4.5             generics_0.0.2
## [37] farver_2.0.3            jsonlite_1.7.1          statnet.common_4.4.1
## [40] crosstalk_1.1.0.1       zip_2.1.1               dendextend_1.14.0
## [43] car_3.0-10              magrittr_1.5            Matrix_1.2-18
## [46] Rcpp_1.0.5              munsell_0.5.0           abind_1.4-5
## [49] viridis_0.5.1           lifecycle_0.2.0         stringi_1.5.3
## [52] yaml_2.2.1              carData_3.0-4           plyr_1.8.6
## [55] grid_4.0.3              promises_1.1.1          forcats_0.5.0
## [58] crayon_1.3.4            miniUI_0.1.1.1          lattice_0.20-41
## [61] haven_2.3.1             splines_4.0.3           hms_0.5.3
## [64] circlize_0.4.10         sna_2.6                 knitr_1.30
## [67] pillar_1.4.6            igraph_1.2.6            ggsignif_0.6.0
## [70] reshape2_1.4.4          rle_0.9.2               glue_1.4.2
## [73] evaluate_0.14           pcaMethods_1.82.0       data.table_1.13.2
## [76] BiocManager_1.30.10     vctrs_0.3.4             httpuv_1.5.4
## [79] cellranger_1.1.0        gtable_0.3.0            purrr_0.3.4
## [82] tidyr_1.1.2             reshape_0.8.8           openxlsx_4.2.3
## [85] xfun_0.18               mime_0.9                xtable_1.8-4
## [88] broom_0.7.2             coda_0.19-4             rstatix_0.6.0
## [91] later_1.1.0.1           class_7.3-17            viridisLite_0.3.0
## [94] tibble_3.0.4            pheatmap_1.0.12         rgl_0.100.54
## [97] ellipsis_0.3.1
```