

Qualitative and multivariate statistical analysis of ligand screening HX-MS experiment

Introduction

Hydrogen-deuterium exchange (HX) is an experiment performed using nuclear magnetic resonance (NMR) spectroscopy or mass spectrometry (MS) to study the protein structure and dynamic. NMR or MS instruments are used to monitor the rate at which the backbone amide hydrogens of the protein are exchanged for the deuterium when protein is incubated in the heavy water (D₂O). In recent years, MS based HX (HX-MS) experiments have been extensively relied on to study the large protein systems that are difficult to probe using existing NMR instruments.¹

In a typical HX-MS experiment workflow, deuterium labelled protein sample is digested using proteolytic enzymes to produce few hundreds of peptides, which are analytically separated and analyzed using MS to calculate the deuterium incorporation on each peptide. This technique is heavily used in the ligand/drug binding studies, where HX properties of protein is studied under unbound and bound states to make inferences regarding binding sites, as well as changes in the dynamics of the protein when the ligand is bound.² Sophisticated software tools are required to validate and interpret data obtained from these experiment due to the high-density and redundancy in the data.²

MS Studio, a software developed by Schriemer Lab at the University of Calgary, is equipped to perform peptide level analysis of HX-MS data.³ In protein-ligand analysis, software such as MS Studio provides two-way comparison of the control and ligand bound states by comparing the differential deuterium uptake of peptides.³ In order to mine the rich data produced by the high-throughput HX-MS screening experiments involving multiple ligands, a multivariate statistical analysis should be employed for the characterization of functionally significant peptides. For such statistical analysis, an R-script is presented in this paper, that can be used as

extension to the MS Studio functionality. The script allows user to plot the %relative deuteration data for each peptide analyzed by MS. This facilitates a qualitative analysis of the data obtained from the HX experiments. In addition, script performs PCA on the peptide deuteration data using “FactoMineR” package, and generates scores and loadings plot to identify peptides that show functionally significant shift in deuterium uptake.⁴ The script can, and will be further developed to implement more sophisticated statistical analysis method such as supervised approach (eg. OPLS-DA). The validated files submitted with the script are from the ligand screening HX-MS experiments performed on the mitotic kinesin Eg5 protein. There were 18 different ligands used in this HX-MS screening study.

1.1 Saving the folders containing data files

Prior to running the R-script, the folders containing validated .csv files, and binding assignments must be downloaded and saved in a desired location. A compressed file containing folders “Exports” and “Binding Assignment” is submitted with the R-script. It is recommended that these two folders be saved on the desktop, since the user will be asked to interactively provide path to the folders when running the R-script (Fig. 1). Also, the files generated by this R-script are saved in the “Exports” folder, which can be discarded along with the folders once the script has been tested.

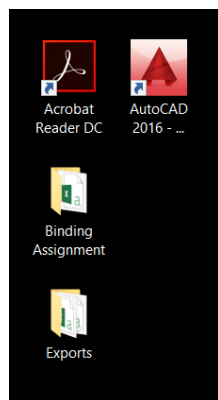


Figure 1. Binding Assignment and Exports folders saved on the desktop.

1.2 Installing and loading packages

The script uses “data.table”, “FactoMineR”, and “ggplot2” packages.⁴⁻⁶ These must be installed and loaded in the environment to run the script successfully. The functionality of the “data.table” package is used, briefly, to aggregate the components of the list elements created in the script.⁶

The “FactoMineR” package is used to conduct principle component analysis(PCA), and “ggplot2” is used to plot the individual and loadings plots by mining the PCA analysis list created by the “FactoMineR”.^{4,5}

Code (1.2)

```
#install.packages("ggplot2")
library(ggplot2)
#install.packages("FactoMineR")
library(FactoMineR)
#install.packages("data.table")
library(data.table)
```

1.3 Setting up working directory and importing the data

Since, this script is used as an extension of the MS Studio functionality, and may be used by the members of the lab that are not familiar with the R language, interactive user input is implemented in the script to provide a path to the working directory and data folder. The user is prompt to provide a path to the “Exports” folder via interactive window using setwd() function (Fig 2). *(NOTE: The interactive window may not pop-up directly on top of the current active window, but appears on the taskbar on the windows desktop. Therefore, user may have to manually select the interactive pop-up on the taskbar to proceed with selection of “Exports” folder).* This path serves dual purpose, one as the working directory where the files generated by the script will be saved, and secondly, it contains the validated data files (.csv) exported from the MS Studio. Once the path to the “Exports” folder is provided, the script proceeds to read and

compile a list of validated files available in the folder. A caption is also included at the top of the interactive window to let the user know what content should be available in the working directory.

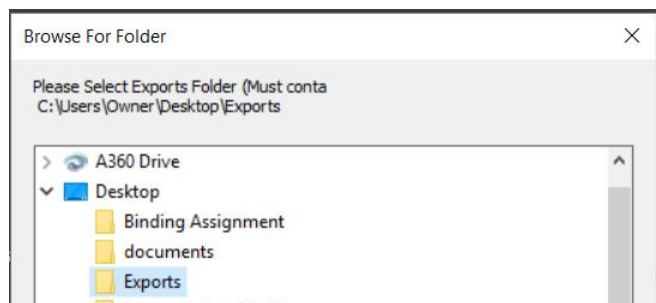


Figure 2. Selecting “Exports” folder

Code (1.3)

```
setwd(choose.dir(caption = "Please Select Exports Folder (Must contain all  
the validated .csv files to be used for further analysis, and no other)"))  
#Prompts user to choose the “Exports” folder  
  
Working_dir <- getwd() #Path to “Exports” folder  
  
temp_files <- list.files(path = Working_dir , pattern = "*.csv") #Compiles  
names of .csv files stored in the “Exports” folder  
  
temp_list <- lapply(temp_files, function(x){read.csv(file = x, header =  
TRUE)}) #Simultaneously reads and stores all the .csv files (as data frames)  
in a list from the “Exports” folder
```

1.4 Extracting and modifying peptide deuteration data for further analysis

The compiled list of data frames from the “Exports” folder contains large amount of extraneous information that is not needed for creating the deuteration plots or performing PCA. Therefore, the data frames of the compiled list are mined simultaneously to extract the first(start) and last(stop) residue number, and charge(z) of each peptide, as well as %corrected fitted deuteration values for each replicate experiments using lapply function and saved as a new list of data frames. The data frames are ordered according to the replicate column, and it is converted from

factor to character format to facilitate the modification of the entries down the road. The start, stop, and z columns of these list of data frames are combined to form a peptide identity column (start_stop_z). The final output provides list of data frames with replicate, peptide identity, and %corrected fitted deuteration columns.

Code (1.4)

```
Trimmed_list <- lapply(temp_list, "[", c("Replicate", "Start", "Stop", "z",  
"Percent.Corrected.Fitted.Deuteration" ))  
#Five columns from each data frames in the list are extracted and stored in a  
new list  
  
Ordered_list <- lapply(Trimmed_list, function(x) x[order(x["Replicate"],  
decreasing = FALSE ), ]) #All elements of the new list are ordered according  
to the "Replicate" column  
  
Combined_ordered_list <- lapply(Ordered_list, function(x) within(x,  
start_stop_z <- paste(x$"Start", x$"Stop", x$"z", sep = "-"))) #Combine  
"Start", "Stop", and "z" columns of the ordered_list and add it to the  
existing list  
  
Combined_char_ordered_list <- lapply(Combined_ordered_list, function(x)  
within(x, Replicate_Char <- paste(x$"Replicate"))) #Adds column of replicates  
(character) to the existing list  
  
Working_list_long <- lapply(Combined_char_ordered_list, "[",  
c("Replicate_Char", "start_stop_z", "Percent.Corrected.Fitted.Deuteration"))  
#List of data frames with replicate(character), start_stop_z, and %corrected  
fitted deuteration columns (long format)
```

1.5 Creating matrix of deuteration values with replicates(sample) and peptide identity as matrix coordinates

As shown in figure 3, long format data frames obtained by running code (1.4) are converted to a wide format, where deuteration values obtained for each replicate samples are laid out by the corresponding peptide identity columns. This modification provides a convenient way of combining the rows of the data frames in the list to create a “working matrix”, which contains deuteration values that can be identified based on the corresponding replicate sample and peptide(start_stop_z).


Replicate_Char	start_stop_z	Percent.Corrected.Fitted.Deuteration		Sample	X13.24.2	X13.24.3	X16.24.2
Data\\THR_001.wiff	13-24-2	22.460614		Data\\THR_001.wiff	22.46061	22.12722	16.63842
Data\\THR_001.wiff	13-24-3	22.127217		Data\\THR_002.wiff	22.32194	22.96428	15.34390
Data\\THR_001.wiff	16-24-2	16.638420		Data\\THR_003.wiff	22.74448	22.55920	15.56312
Data\\THR_001.wiff	18-24-2	6.112084		Data\\THR_004.wiff	19.57778	19.56786	14.19570
Data\\THR_001.wiff	25-30-1	60.384321					
Data\\THR_001.wiff	25-30-2	62.229944					
Data\\THR_001.wiff	31-48-4	38.196459					
Data\\THR_001.wiff	31-49-4	35.957467					
Data\\THR_001.wiff	34-44-2	33.878457					

Figure 3. Long format data frame obtained by running code (1.4) is converted to the wide format by running code (1.5). Only the first few rows of the long format and the first few columns of the wide format are shown here.

Code (1.5)

```
Working_list_wide <- lapply(Working_list_long, function(x)
data.frame(matrix(data = x$"Percent.Corrected.Fitted.Deuteration", nrow =
length(unique(x$"Replicate_Char")), ncol = length(unique(x$"start_stop_z")),
byrow = TRUE), row.names = unique(x$"Replicate_Char"))) #converts list of
data frames to wide format
```

```
Transposed_start_stop_z <- lapply(Working_list_long, function(x)
t(data.frame(c(unique(x$"start_stop_z"))))) #Transposes the peptide identity
column(start_stop_z)
```

```
for (i in 1:length(Working_list_wide)){
  names(Working_list_wide[[i]]) <- Transposed_start_stop_z[[i]]
} #Loops through list of transposed start_stop_z column and assigns them as
column names of the wide format list
```

```
Additional_rep_column <- lapply(Working_list_wide, function(x)
data.frame(row.names(x)))
for(i in 1:length(Additional_rep_column)){
  names(Additional_rep_column[[i]]) <- c("Sample")
} #creates of a list of replicate columns from wide format and renames every
column as "Sample"
```

```
Working_list_complete <- list()
for(i in 1:length(Additional_rep_column)){
  Working_list_complete[[i]] <- data.frame(cbind(Working_list_wide[[i]],
Additional_rep_column[[i]]))
} #combines wide format data frames and sample column
```

```
Working_matrix <- rbindlist(lapply(Working_list_complete,
```

```
as.data.frame.list), fill = TRUE) #binds rows of data frames in a list to
provide a matrix containing all the deuteration data
```

1.6 Modifying sample entries and assigning class variables

Sample column entries of the “working matrix” is stripped of extraneous characters (Fig. 4). In addition, the class variable is assigned to each row, by simply removing the replicate numbering from the sample entries (Fig. 4). The class variable is used for the binding or non-binding assignment for each sample in the matrix.



Figure 4. Modification of character string entries of sample column to obtain “Sample” (1.6.1) and “Class” (1.6.2) entries.

Code (1.6.1)

```
Class_column <- Working_matrix[, "Sample"]
Class_column$Sample <- as.character(Class_column$Sample)
Class_column$Sample <- sub("[^:alnum:]", "", Class_column$Sample,
ignore.case = TRUE)
Class_column$Sample <- sub("Data", "", Class_column$Sample, ignore.case =
TRUE)
Class_column$Sample <- sub(".wiff", "", Class_column$Sample, ignore.case =
TRUE)
Class_column$Sample <- sub("_", "", Class_column$Sample, ignore.case = TRUE)
Replicate_column <- Class_column
#This chunk of code modifies the “Sample” column of the working matrix by
removing the extraneous characters from every entry.
```

Code (1.6.2)

```
Class_column$Sample <- substr(Class_column$Sample, 1,
nchar(Class_column$Sample)-3) #Removes numbers from sample entries

Working_matrix <- subset(Working_matrix, select = -c(Sample)) #Removes sample
column with extraneous characters
```

```
Working_matrix <- cbind(Class_column, Working_matrix) #binds class variable to working matrix

colnames(Working_matrix) <- sub(colnames(Working_matrix[,1]), "Class",
colnames(Working_matrix), ignore.case = TRUE) #column name is changed from "Sample" to "Class"
Working_matrix <- cbind(Replicate_column, Working_matrix) #binds "Sample" column without extraneous characters to the working matrix
```

1.7 Assessing class variables and assigning binding or non-binding designation to class variables

User is interactively asked to provide a path to “Binding_assignment.csv” (Fig. 5). The binding assignment file contains information regarding the class of each sample, and corresponding binding or non-binding designation to be used for the analysis. Based on the information available in the file, the class variable column in the working matrix is modified. Although optional, this operation ensures that if a particular sample in the list is assigned a class that is different than the sample name, it is changed accordingly. If no change in the class entries detected, the code still runs, but does not make any changes to the class column of the working matrix.

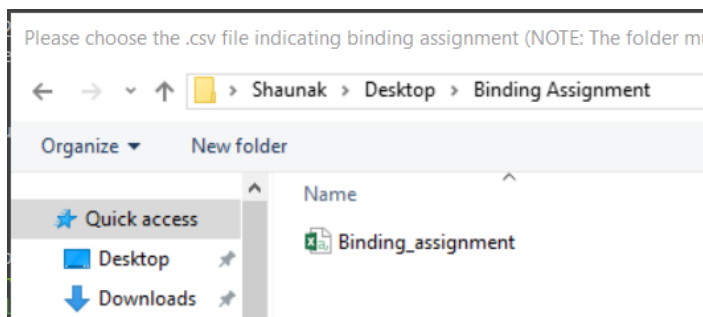


Figure 5. Selection of binding assignment

Code (1.7)

```
temp_control_files <- choose.files(caption = "Please choose the .csv file
indicating binding assignment (NOTE: The folder must not contain any
additional .csv files)") #User is asked to provide location of the
"Binding_Assignment.csv".

temp_control_df <- as.data.frame(read.csv(paste(gsub("\\", "/",
```



```
temp_control_files, fixed =TRUE)), header = TRUE, stringsAsFactors = FALSE))
#reads binding assignment file

merged_column <- merge(Class_column, temp_control_df, by = "Sample", all=
TRUE) #Class column of the working matrix is merged with the binding
assignment by sample column

Working_matrix[, "Class"] <- merged_column[, "Class"] #Class column is replaced
with the new class column based on the information from binding assignment
file

Working_matrix <- cbind(merged_column[, "Binding.Non.binding"],
Working_matrix) #Binding or non-binding designation is added to the working
matrix

Working_matrix <-
Working_matrix[order(Working_matrix[, "Binding.Non.binding"])] #working matrix
is ordered according to the binding/non-binding column
Working_matrix <- as.data.frame(Working_matrix) #working matrix was created
using rbindlist() function from the data.table package. In order to avoid any
problems in the subsequent code, it is reverted back to the data frame format

Working_matrix[Working_matrix <= 0] <- NA #missing values in the working
matrix are assigned NA
```

1.8 Removing missing data and some formatting

Peptide column containing more than 50% NAs are removed from the working matrix. The column names (peptide identities) are converted to the “start-stop-z+” format, which will be used to identify the deuteration plots. The average value for the %deuteration is calculated by class variable and saved as a data table. The data frame with individual values (working matrix) and the average values (grouped matrix) of %deuteration are saved in the “Exports” folder.

Code (1.8)

```
limit_for_NA <- round(nrow(Working_matrix)/2) #value equal to the half the
number of rows in working matrix

Working_matrix <- Working_matrix[ ,
which(as.numeric(colSums(!is.na(Working_matrix)))>limit_for_NA)] #This chunk
of code removes columns of the working matrix that contain 50% or more NAs
```

```

Working_matrix$Sample <- sub("00", "_", Working_matrix$Sample)
pep_ions <- colnames(Working_matrix)[c(4:length(colnames(Working_matrix)))]
pep_ions <- sub("X", "", pep_ions)
pep_ions <- gsub("[:punct:]", "-", pep_ions)
pep_ions <- paste(pep_ions, "+", sep = "")
colnames(Working_matrix)[c(4:length(colnames(Working_matrix)))] <- pep_ions
#This chunk of code modifies the column names (peptide identity) of the
working matrix by converting them to the start-stop-z+ format.

Working_matrix[sapply(Working_matrix, is.character)] <-
lapply(Working_matrix[sapply(Working_matrix, is.character)], as.factor)
#converts character vectors in the working matrix to factors

Working_matrix_dt <- data.table(Working_matrix)
Grouped_matrix <- Working_matrix_dt[, lapply(.SD, mean), by = "Class",
.SDcols = names(Working_matrix_dt)[4:ncol(Working_matrix_dt)]] #calculates
average %deuterium incorporated for each peptide by class and creates a new
matrix of these average value

dir.create("Condensed matrices") #creates a directory in the "Exports" folder
to save the matrices
write.csv(Working_matrix, file = paste(Working_dir, "/", "Condensed
matrices", "/", "Workingmatrix.csv", sep = ""), row.names = FALSE)
write.csv(Grouped_matrix, file = paste(Working_dir, "/", "Condensed
matrices", "/", "Groupedmatrix.csv", sep = ""), row.names = FALSE)

```

1.9 Deuteration and PCA plots

Deuteration plots are created from the “working matrix”, using “ggplot2” package, by looping through each column (peptide) and plotting the corresponding %relative deuteration vs. sample (Fig. 6). This for loop also saves the individual plots in the working directory. It takes approximately 1 minute to generate and save all the deuteration plots. In addition, the PCA was performed on the same deuteration data using the PCA() function of the “FactoMineR” package. Prior to performing the PCA analysis data was stripped of all the peptide columns containing NAs. The “FactoMineR” package provides a convenient list of the PCA data, from which relevant data for the scores and loadings plots were extracted. First five scores and loadings components were plotted and saved in the working directory (Fig. 7 and 8).

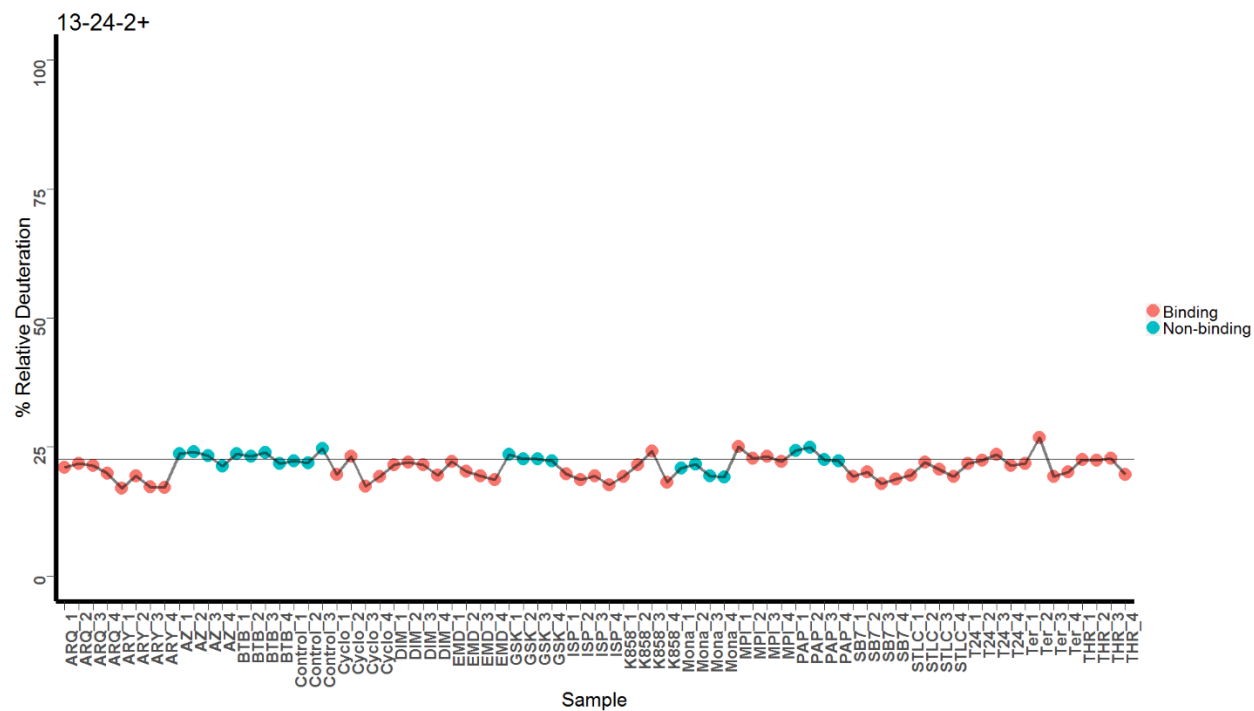


Figure 6. %relative deuteration plot for peptide 13-24-2+.

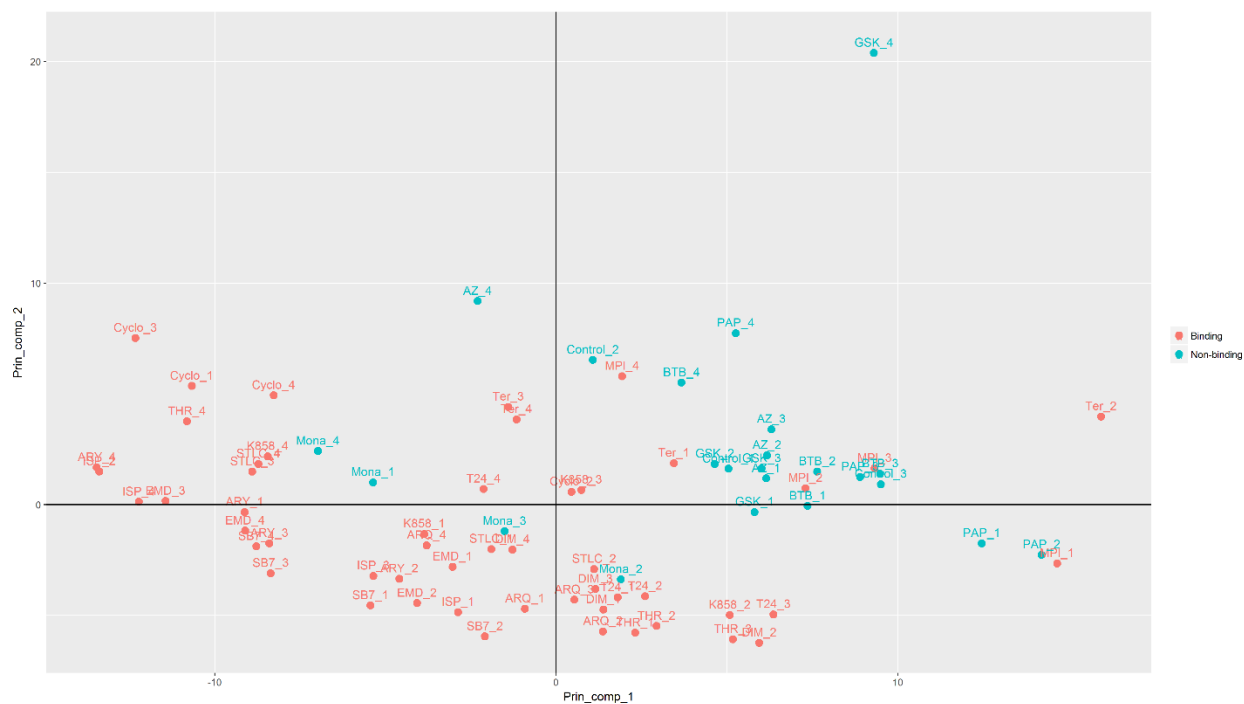


Figure 7. Scores plot.

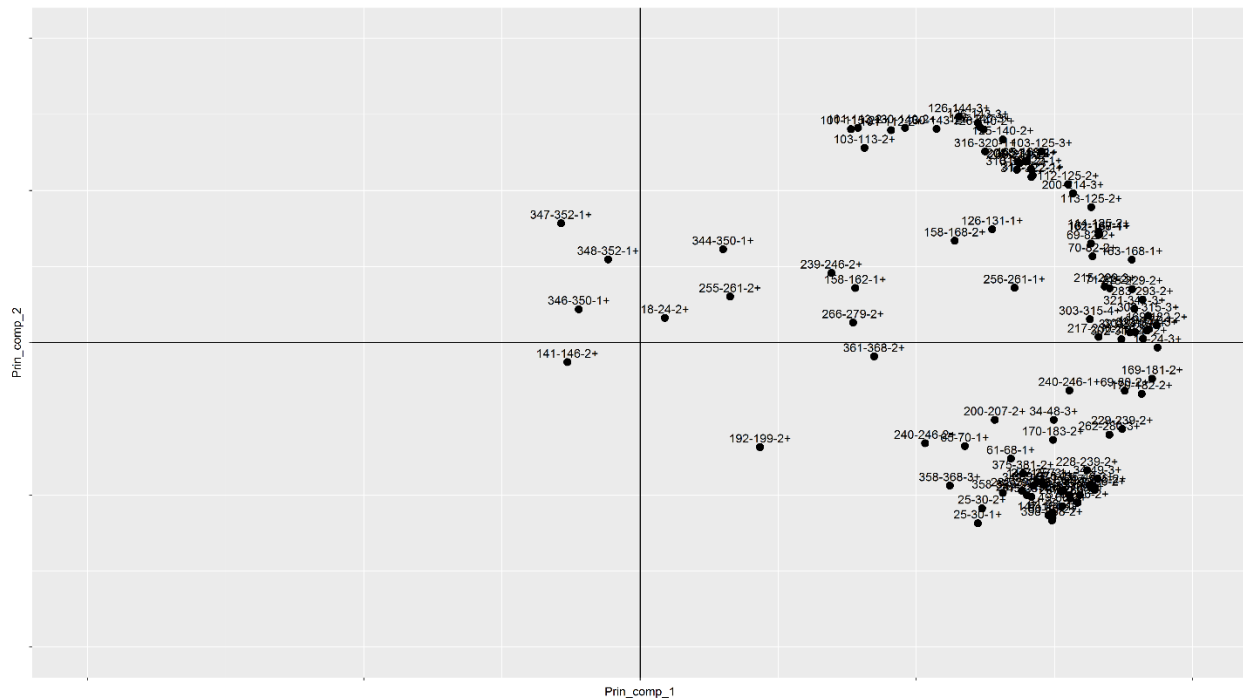


Figure 8. Loadings plot.

Code (1.9)

```
##deuteration plots##

dir.create("Deuteration plots")#creates a directory in the "Exports" folder to
o save the deuteration plots

for(i in 4:ncol(Working_matrix))
{
  pathname <- file.path(paste(Working_dir, "/", "Deuteration plots", "/", col
names(Working_matrix[i]), ".tiff", sep = "")) #creates path to the working di
rectory to save the deuteration plots with corresponding peptide title
  tiff(file = pathname, width = 1920, height = 1080) #file format and resolut
ion

print(ggplot(Working_matrix, aes(Sample, Working_matrix[,i], fill = Class)) +
  ggtitle(colnames(Working_matrix[i])) +
  geom_point(aes(color = Binding.Non.binding), size = 9, na.rm = TRUE) +
  geom_line(color = "black", group = 1, size = 1.5, stat = "density", alpha =
0.5, na.rm = TRUE)+
  ylab("% Relative Deuteration") +
  ylim(0,100) +
  geom_hline(yintercept = mean(Working_matrix[Working_matrix$Class == "Contro
l", i], na.rm = TRUE)) +
```

```

    theme(legend.position = "right", legend.title = element_blank(), title = el
element_text(color = "black", size = 30), axis.text = element_text(angle = 90,
hjust = 1, vjust = 0.5, size = 25, face = "bold"), panel.background = element
_rect(fill = "white", color = "white"), axis.line = element_line(color = "bla
ck", size = 2.5), legend.text = element_text(size = 25), axis.ticks.length =
unit(0.5, "cm")) +
    guides(alpha = FALSE, fill = FALSE))

dev.off() #closes the plotting device
} #This chunk of code creates deuteration plots (%relative deuteration vs. sa
mple) by looping through the columns of the working matrix. One plot for each
peptide is created and saved as a high resolution tiff file in the "Exports"
folder. Each plot has a horizontal line indicating the mean value for the non
-binding class of ligand. Color of the data points also indicate binding or n
on-binding assignment for each sample.

##PCA##

working_matrix_PCA_no_NA <- Working_matrix[ , colSums(is.na(Working_matrix))
== 0] #columns containing NAs are removed from the working matrix prior to pe
rforming PCA
Working_PCA <- PCA(working_matrix_PCA_no_NA[-c(1:3)], graph = FALSE) #PCA() f
unction from FactoMineR package is used to perform PCA

dir.create("Scores plots") #directory is created to save the Scores plots
Ind_prin_comp_1 <- Working_PCA$ind$coord[,1]
Ind_prin_comp_2 <- Working_PCA$ind$coord[,2]
Ind_prin_comp_3 <- Working_PCA$ind$coord[,3]
Ind_prin_comp_4 <- Working_PCA$ind$coord[,4]
Ind_prin_comp_5 <- Working_PCA$ind$coord[,5] #first five scores

Working_PCA_plot_matrix <- as.data.frame(cbind(Working_matrix[c(1:3)], Ind_pr
in_comp_1, Ind_prin_comp_2, Ind_prin_comp_3, Ind_prin_comp_4, Ind_prin_comp_5
)) #data frame containing sample, class, binding assignment and, five scores

for(i in 4:7)
{
  ggplot(Working_PCA_plot_matrix, aes(Working_PCA_plot_matrix[,i], Working_PCA_
plot_matrix[,i+1], color = Binding.Non.binding)) +
    ylab(paste("Prin_comp_", i-2, sep = "")) +
    xlab(paste("Prin_comp_", i-3, sep = "")) +
    geom_point(size = 3) +
    geom_text(aes(label = Sample), nudge_y = 0.5) +
    geom_hline(yintercept = 0) +
    geom_vline(xintercept = 0) +
    theme(legend.title = element_blank())
  ggsave(filename = paste("PCA_Ind", "-", i-3, ".tiff", sep = ""), path = paste(

```

```

Working_dir, "/", "Scores plots", sep = ""), dpi = 300, height = 9, width = 1
6, units = c("in"))
} #This chunk of code creates scores plots and saves them in the "Exports" fo
lder

dir.create("Loadings plots") #directory is created to save the Loadings plots
Var_prin_comp_1 <- as.numeric(Working_PCA$var$coord[,1])
Var_prin_comp_2 <- as.numeric(Working_PCA$var$coord[,2])
Var_prin_comp_3 <- as.numeric(Working_PCA$var$coord[,3])
Var_prin_comp_4 <- as.numeric(Working_PCA$var$coord[,4])
Var_prin_comp_5 <- as.numeric(Working_PCA$var$coord[,5]) #first five Loadings
Ion <- colnames(working_matrix_PCA_no_NA[-c(1:3)])

Working_PCA_plot_matrix_2 <- data.frame(Ion, Var_prin_comp_1, Var_prin_comp_2
, Var_prin_comp_3, Var_prin_comp_4, Var_prin_comp_5, row.names = NULL) #data
frame containing peptide identity column, and first five Loadings

for(i in 2:5)
{
ggplot(Working_PCA_plot_matrix_2, aes(Working_PCA_plot_matrix_2[,i], Working_
PCA_plot_matrix_2[,i+1])) +
  ylab(paste("Prin_comp_", i, sep = "")) +
  xlab(paste("Prin_comp_", i-1, sep = "")) +
  geom_point(size = 3) +
  geom_text(aes(label = Ion), nudge_y = 0.03) +
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = 0) +
  ylim(-1,1)+
  xlim(-1,1)+
  theme(legend.title = element_blank(), axis.text.x = element_blank(), axis.t
ext.y = element_blank())
ggsave(filename = paste("PCA_Var", "-", i-1, ".tiff", sep = ""), path = paste
(Working_dir, "/", "Loadings plots", sep = ""), dpi = 300, height = 9, width
= 16, units = c("in"))
} #This chunk of code creates Loadings plots and saves them in the "Exports"
folder

```

Complete R-script

```
###1.2###

#install.packages("ggplot2")
library(ggplot2)
#install.packages("FactoMineR")
library(FactoMineR)
#install.packages("data.table")
library(data.table)

###1.3###

setwd(choose.dir(caption = "Please Select Exports Folder (Must contain all the validated .csv files to be used for further analysis, and no other)")) #Prompts user to choose the "Exports" folder
Working_dir <- getwd() #Path to "Exports" folder
temp_files <- list.files(path = Working_dir, pattern = "*.csv") #Compiles names of .csv files stored in the "Exports" folder
temp_list <- lapply(temp_files, function(x){read.csv(file = x, header = TRUE)}) #Simultaneously reads and stores all the .csv files (as data frames) in a list from the "Exports" folder

###1.4###

Trimmed_list <- lapply(temp_list, "[", c("Replicate", "Start", "Stop", "z", "Percent.Corrected.Fitted.Deuteration")) #Five columns from each data frames in the list are extracted and stored in a new list

Ordered_list <- lapply(Trimmed_list, function(x) x[order(x["Replicate"]), decreasing = FALSE], []) #All elements of the new list are ordered according to the "Replicate" column

Combined_ordered_list <- lapply(Ordered_list, function(x) within(x, start_stop_z <- paste(x$"Start", x$"Stop", x$"z", sep = "-")))#Combine "Start", "Stop", and "z" columns of the ordered list and add it to the existing list

Combined_char_ordered_list <- lapply(Combined_ordered_list, function(x) within(x, Replicate_Char <- paste(x$"Replicate")) #Adds column of replicate(character) to the existing list

Working_list_long <- lapply(Combined_char_ordered_list, "[", c("Replicate_Char", "start_stop_z", "Percent.Corrected.Fitted.Deuteration"))#List of data frames with replicate(character), start_stop_z, and %corrected fitted deuteration columns (long format)
```

```
###1.5###
```

```
Working_list_wide <- lapply(Working_list_long, function(x) data.frame(matrix(
data = x$"Percent.Corrected.Fitted.Deuteration", nrow = length(unique(x$"Replicate_Char")), ncol = length(unique(x$"start_stop_z")), byrow = TRUE), row.names = unique(x$"Replicate_Char"))) #converts list of data frames to wide format
```

```
Transposed_start_stop_z <- lapply(Working_list_long, function(x) t(data.frame(
(c(unique(x$"start_stop_z"))))) #Transposes the peptide identity column(start_stop_z)
```

```
for (i in 1:length(Working_list_wide)){
  names(Working_list_wide[[i]]) <- Transposed_start_stop_z[[i]]
} #Loops through list of transposed start_stop_z column and assigns them as column names of the wide format list
```

```
Additional_rep_column <- lapply(Working_list_wide, function(x) data.frame(row.names(x)))
for(i in 1:length(Additional_rep_column)){
  names(Additional_rep_column[[i]]) <- c("Sample")
} #creates of a list of replicate columns from wide format and renames every column as "Sample"
```

```
Working_list_complete <- list()
for(i in 1:length(Additional_rep_column)){
  Working_list_complete[[i]] <- data.frame(cbind(Working_list_wide[[i]], Additional_rep_column[[i]]))
} #combines wide format data frames and sample column
```

```
Working_matrix <- rbindlist(lapply(Working_list_complete, as.data.frame.list), fill = TRUE) #binds rows of data frames in a list to provide a matrix containing all the deuteration data
```

```
###1.6.1###
```

```
Class_column <- Working_matrix[, "Sample"]
Class_column$Sample <- as.character(Class_column$Sample)
Class_column$Sample <- sub("[^[:alnum:]]", "", Class_column$Sample, ignore.case = TRUE)
Class_column$Sample <- sub("Data", "", Class_column$Sample, ignore.case = TRUE)
Class_column$Sample <- sub(".wiff", "", Class_column$Sample, ignore.case = TRUE)
Class_column$Sample <- sub("_", "", Class_column$Sample, ignore.case = TRUE)
```



```
Replicate_column <- Class_column #This chunk of code modifies the "Sample" column of the working matrix by removing the extraneous characters from every entry.
```

```
###1.6.2###
```

```
Class_column$Sample <- substr(Class_column$Sample, 1, nchar(Class_column$Sample)-3) #Removes numbers from sample entries
```

```
Working_matrix <- subset(Working_matrix, select = -c(Sample)) #Removes sample column with extraneous characters
```

```
Working_matrix <- cbind(Class_column, Working_matrix) #binds class variable to working matrix
```

```
colnames(Working_matrix) <- sub(colnames(Working_matrix[,1]), "Class", colnames(Working_matrix), ignore.case = TRUE) #column name is changed from "Sample" to "Class"
```

```
Working_matrix <- cbind(Replicate_column, Working_matrix) #binds "Sample" column without extraneous characters to the working matrix
```

```
###1.7###
```

```
temp_control_files <- choose.files(caption = "Please choose the .csv file indicating binding assignment (NOTE: The folder must not contain any additional .csv files)") #User is asked to provide location of the "Binding_Assignment.csv"
```

```
temp_control_df <- as.data.frame(read.csv(paste(gsub("\\", "/", temp_control_files, fixed = TRUE)), header = TRUE, stringsAsFactors = FALSE)) #reads binding assignment file
```

```
merged_column <- merge(Class_column, temp_control_df, by = "Sample", all= TRUE) #Class column of the working matrix is merged with the binding assignment by sample column
```

```
Working_matrix <- cbind(merged_column[, "Binding.Non.binding"], Working_matrix) #Binding or non-binding designation is added to the working matrix
```

```
Working_matrix[, "Class"] <- merged_column[, "Class"] #Class column is replaced with the new class column based on the information from binding assignment file
```

```
Working_matrix <- Working_matrix[order(Working_matrix[, "Binding.Non.binding"])] #working matrix is ordered according to the binding/non-binding column
```

```
Working_matrix <- as.data.frame(Working_matrix) #working matrix was created using rbindlist() function from the data.table package. In order to avoid any problems in the subsequent code, it is reverted back to the data frame format
```

```
Working_matrix[Working_matrix <= 0] <- NA #missing values in the working matrix are assigned NA
```

```
###1.8###
```

```
limit_for_NA <- round(nrow(Working_matrix)/2) #value equal to the half the number of rows in working matrix
```

```
Working_matrix <- Working_matrix[, which(as.numeric(colSums(!is.na(Working_matrix))) > limit_for_NA)] #This chunk of code removes columns of the working matrix that contain 50% or more NAs
```

```
Working_matrix$Sample <- sub("00", "_", Working_matrix$Sample)
pep_ions <- colnames(Working_matrix)[c(4:length(colnames(Working_matrix)))]
pep_ions <- sub("X", "", pep_ions)
pep_ions <- gsub("[:punct:]", "-", pep_ions)
pep_ions <- paste(pep_ions, "+", sep = "")
colnames(Working_matrix)[c(4:length(colnames(Working_matrix)))] <- pep_ions # This chunk of code modifies the column names (peptide identity) of the working matrix by converting them to the start-stop-z+ format.
```

```
Working_matrix[sapply(Working_matrix, is.character)] <- lapply(Working_matrix[sapply(Working_matrix, is.character)], as.factor) #converts character vectors in the working matrix to factors
```

```
Working_matrix_dt <- data.table(Working_matrix)
Grouped_matrix <- Working_matrix_dt[, lapply(.SD, mean), by = "Class", .SDcols = names(Working_matrix_dt)[4:ncol(Working_matrix_dt)]] #calculates average %deuterium incorporated for each peptide by class and creates a new matrix of these average values
```

```
dir.create("Condensed matrices") #creates a directory in the "Exports" folder to save the matrices
```

```
write.csv(Working_matrix, file = paste(Working_dir, "/", "Condensed matrices", "/", "Workingmatrix.csv", sep = ""), row.names = FALSE)
```

```
write.csv(Grouped_matrix, file = paste(Working_dir, "/", "Condensed matrices", "/", "Groupedmatrix.csv", sep = ""), row.names = FALSE)
```

```

####1.9####

##deuteration plots##

dir.create("Deuteration plots")#creates a directory in the "Exports" folder to
o save the deuteration plots

for(i in 4:ncol(Working_matrix))
{
  pathname <- file.path(paste(Working_dir, "/", "Deuteration plots", "/", col
names(Working_matrix[i]), ".tiff", sep = "")) #creates path to the working di
rectory to save the deuteration plots with corresponding peptide title
  tiff(file = pathname, width = 1920, height = 1080) #file format and resolut
ion

print(ggplot(Working_matrix, aes(Sample, Working_matrix[,i], fill = Class)) +
  ggtitle(colnames(Working_matrix[i])) +
  geom_point(aes(color = Binding.Non.binding), size = 9, na.rm = TRUE) +
  geom_line(color = "black", group = 1, size = 1.5, stat = "density", alpha =
0.5, na.rm = TRUE)+
  ylab("% Relative Deuteration") +
  ylim(0,100) +
  geom_hline(yintercept = mean(Working_matrix[Working_matrix$Class == "Contro
l", i], na.rm = TRUE)) +
  theme(legend.position = "right", legend.title = element_blank(), title = el
ement_text(color = "black", size = 30), axis.text = element_text(angle = 90,
hjust = 1, vjust = 0.5, size = 25, face = "bold"), panel.background = element
_rect(fill = "white", color = "white"), axis.line = element_line(color = "bla
ck", size = 2.5), legend.text = element_text(size = 25), axis.ticks.length =
unit(0.5, "cm")) +
  guides(alpha = FALSE, fill = FALSE))

dev.off() #closes the plotting device
} #This chunk of code creates deuteration plots (%relative deuteration vs. sa
mple) by looping through the columns of the working matrix. One plot for each
peptide is created and saved as a high resolution tiff file in the "Exports"
folder. Each plot has the a horizontal line indicating the mean value for the
non-binding class of ligand. Color of the data points also indicate binding o
r non-binding assignment for each sample.

##PCA##

working_matrix_PCA_no_NA <- Working_matrix[, colSums(is.na(Working_matrix))
== 0] #columns containing NAs are removed from the working matrix prior to pe
rforming PCA
Working_PCA <- PCA(working_matrix_PCA_no_NA[-c(1:3)], graph = FALSE) #PCA() f
unction from FactoMineR package is used to perform PCA

```

```

dir.create("Scores plots") #directory is created to save the Scores plots
Ind_prin_comp_1 <- Working_PCA$ind$coord[,1]
Ind_prin_comp_2 <- Working_PCA$ind$coord[,2]
Ind_prin_comp_3 <- Working_PCA$ind$coord[,3]
Ind_prin_comp_4 <- Working_PCA$ind$coord[,4]
Ind_prin_comp_5 <- Working_PCA$ind$coord[,5] #first five scores

Working_PCA_plot_matrix <- as.data.frame(cbind(Working_matrix[c(1:3)], Ind_pr
in_comp_1, Ind_prin_comp_2, Ind_prin_comp_3, Ind_prin_comp_4, Ind_prin_comp_5
)) #data frame containing sample, class, binding assignment and, five scores

for(i in 4:7)
{
ggplot(Working_PCA_plot_matrix, aes(Working_PCA_plot_matrix[,i], Working_PCA_
plot_matrix[,i+1], color = Binding.Non.binding)) +
  ylab(paste("Prin_comp_", i-2, sep = "")) +
  xlab(paste("Prin_comp_", i-3, sep = "")) +
  geom_point(size = 3) +
  geom_text(aes(label = Sample), nudge_y = 0.5) +
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = 0) +
  theme(legend.title = element_blank())
ggsave(filename = paste("PCA_Ind","-", i-3, ".tiff", sep = ""), path = paste(
Working_dir, "/", "Scores plots", sep = ""), dpi = 300, height = 9, width = 1
6, units = c("in"))
} #This chunk of code creates scores plots and saves them in the "Exports" fo
lder

dir.create("Loadings plots") #directory is created to save the Loadings plots
Var_prin_comp_1 <- as.numeric(Working_PCA$var$coord[,1])
Var_prin_comp_2 <- as.numeric(Working_PCA$var$coord[,2])
Var_prin_comp_3 <- as.numeric(Working_PCA$var$coord[,3])
Var_prin_comp_4 <- as.numeric(Working_PCA$var$coord[,4])
Var_prin_comp_5 <- as.numeric(Working_PCA$var$coord[,5]) #first five Loadings
Ion <- colnames(working_matrix_PCA_no_NA[-c(1:3)])

Working_PCA_plot_matrix_2 <- data.frame(Ion, Var_prin_comp_1, Var_prin_comp_2
, Var_prin_comp_3, Var_prin_comp_4, Var_prin_comp_5, row.names = NULL) #data
frame containing peptide identity column, and first five Loadings

for(i in 2:5)
{
ggplot(Working_PCA_plot_matrix_2, aes(Working_PCA_plot_matrix_2[,i], Working_
PCA_plot_matrix_2[,i+1])) +
  ylab(paste("Prin_comp_", i, sep = "")) +
  xlab(paste("Prin_comp_", i-1, sep = "")) +

```

```

geom_point(size = 3) +
geom_text(aes(label = Ion), nudge_y = 0.03) +
geom_hline(yintercept = 0) +
geom_vline(xintercept = 0) +
ylim(-1,1)+
xlim(-1,1)+
theme(legend.title = element_blank(), axis.text.x = element_blank(), axis.t
ext.y = element_blank())
ggsave(filename = paste("PCA_Var", "-", i-1, ".tiff", sep = ""), path = paste
(Working_dir, "/", "Loadings plots", sep = ""), dpi = 300, height = 9, width
= 16, units = c("in"))
} #This chunk of code creates loadings plots and saves them in the "Exports"
folder

sessionInfo()

## R version 3.3.3 (2017-03-06)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 14393)
##
## locale:
## [1] LC_COLLATE=English_Canada.1252 LC_CTYPE=English_Canada.1252
## [3] LC_MONETARY=English_Canada.1252 LC_NUMERIC=C
## [5] LC_TIME=English_Canada.1252
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## other attached packages:
## [1] data.table_1.10.4 FactoMineR_1.35  ggplot2_2.2.1
##
## loaded via a namespace (and not attached):
## [1] Rcpp_0.12.8      cluster_2.0.5    knitr_1.15.1
## [4] magrittr_1.5     leaps_3.0        MASS_7.3-45
## [7] scatterplot3d_0.3-38 munsell_0.4.3    colorspace_1.3-2
## [10] lattice_0.20-34  stringr_1.1.0    plyr_1.8.4
## [13] tools_3.3.3      grid_3.3.3       gtable_0.2.0
## [16] htmltools_0.3.5  yaml_2.1.14      lazyeval_0.2.0
## [19] rprojroot_1.1    digest_0.6.11    assertthat_0.1
## [22] tibble_1.2       evaluate_0.10    rmarkdown_1.3
## [25] labeling_0.3     stringi_1.1.2    scales_0.4.1
## [28] backports_1.0.4  flashClust_1.01-2

```

References

1. Percy AJ, Rey M, Burns KM, Schriemer DC: **Probing protein interactions with hydrogen/deuterium exchange and mass spectrometry—A review**. *Anal Chim Acta* 2012, **721**:7–21.
2. Konermann L, Pan J, Liu Y-H, Fadgen KE, Brown J, Engen JR, Griffin PR, Chalmers MJ, Kamenecka TM, Blüher M, Griffin PR, Spiegelman B: **Hydrogen exchange mass spectrometry for studying protein structure and dynamics**. *Chem Soc Rev* 2011, **40**:1224–1234.
3. Rey M, Sarpe V, Burns KM, Buse J, Baker CAH, van Dijk M, Wordeman L, Bonvin AMJJ, Schriemer DC: **Mass Spec Studio for Integrative Structural Biology**. *Structure* 2014, **22**:1538–1548.
4. Lê S, Josse J, Husson F: **FactoMineR : An R Package for Multivariate Analysis**. *J Stat Softw* 2008, **25**:1–18.
5. Wickham H: **Ggplot2 : Elegant Graphics for Data Analysis**. Springer-Verlag New York, 2009.
6. Matt Dowle, Arun Srinivasan. **Data.table: Extension of ‘data.frame’**. 2017