An introduction to the PepSAVIms package

November 21, 2024

Contents

0 Introduction **1**

1 Package Overview **2**
1.1 The `binMS` function . . . 2
1.2 The `filterMS` function . . . 3
1.3 The `rankEN` function . . . 3
1.4 The `msDat` function . . . 4

2 An example case: performing the PepSAVI-MS pipeline **4**
2.1 Reading in the data . . . 4
2.1.1 Reading in the mass spectrometry data . . . 4
2.1.2 Reading in the bioactivity data . . . 5
2.2 Using `binMS` . . . 6
2.2.1 `binMS` specification via variable names . . . 7
2.2.2 `binMS` via names, indices, and vectors . . . 7
2.2.3 The print and summary function for `binMS` . . . 8
2.3 Using `filterMS` . . . 9
2.3.1 `filterMS`: specifying the region of interest . . . 10
2.3.2 `filterMS`: specifying bordering region . . . 10
2.3.3 The print and summary function for `filterMS` . . . 11
2.4 Using `rankEN` . . . 12
2.4.1 `rankEN`: specifying the region of interest for mass spectrometry and bioactivity data . . . 12
2.4.2 The summary function for `rankEN` . . . 12
2.4.3 Accessing the ranked candidate compounds . . . 14

3 Data access and manipulation tools **14**
3.1 The mass spectrometry data class hierarchy . . . 14
3.2 The `extractMS` function . . . 15
3.2.1 Extracting a `matrix` object . . . 15
3.2.2 Extracting an `msDat` object . . . 16
3.3 The `msDat` function . . . 16
3.4 The `msDat` class interface . . . 17

0 Introduction

The PepSAVIms R package provides a collection of software tools used to facilitate the prioritization of putative bioactive compounds from a complex biological matrix. The package was constructed to provide an implementation of the statistical portion of the laboratory and statistical procedure proposed in *The PepSAVI-MS pipeline for natural product bioactive peptide discovery* (herafter abbreviated to *The PepSAVI-MS pipeline*) [**?**].

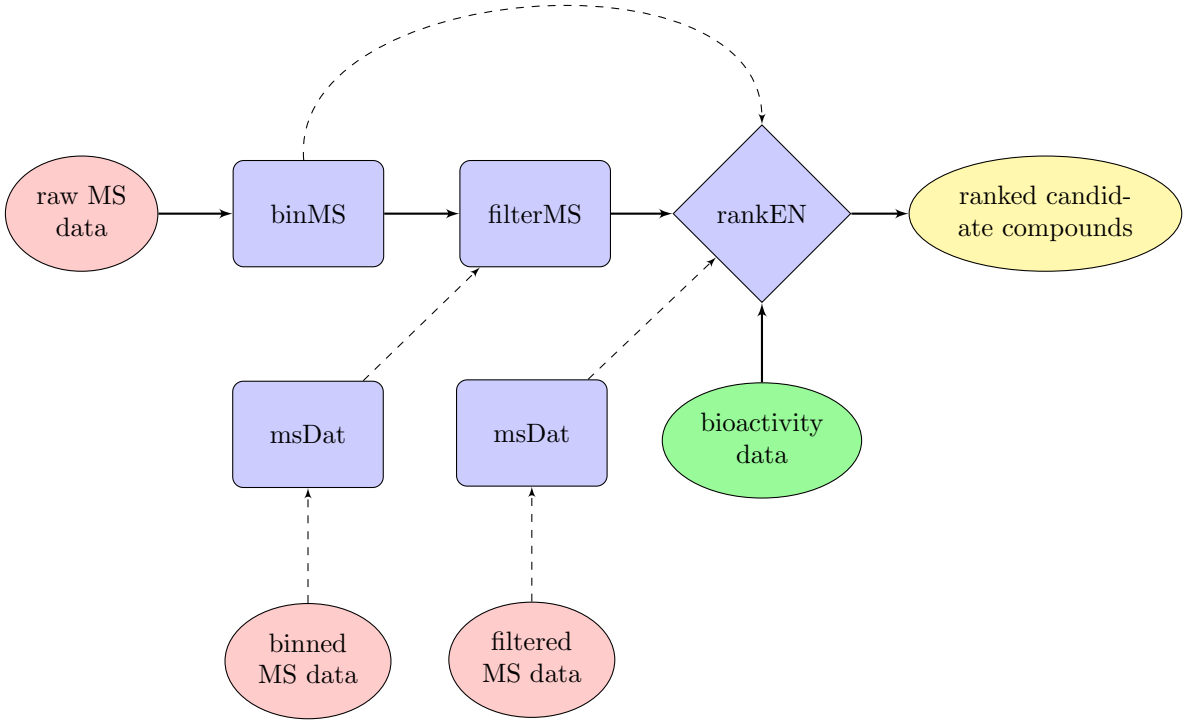
This document provides an introduction to the functionality provided by the PepSAVIms package.

1 Package Overview

A flowchart for a data analysis performed using PepSAVIms is show in Figure 1. The blue rectangles and diamond are functions in the PepSAVIms ecosystem; the pink oval denotes mass spectrometry data to be used as data inputs; the green oval denotes bioactivity data to be used as a data input; and the yellow oval denotes the analysis results of using the PepSAVI-MS methodology and software.

The prototypical data analysis workflow is the path described by the solid lines; this is the procedure performed in *The PepSAVI-MS pipeline*, and described in more detail in the `Paper_Analysis` vignette. The dashed lines show alternative workflows which may be performed in situations where some of the data processing steps have already been performed using other tools, or where some of the data processing steps may not be appropriate for the particular data analysis at hand.

Figure 1: Data analysis flow chart



Solid arrows represent the prototypical data analysis workflow;
dashed lines represent alternative workflows

A conceptual presentation of the functions in the PepSAVIms package is provided in the following subsections. Please refer to the function documentation for more information regarding the application programming interface, as well as for further technical detail.

1.1 The `binMS` function

The mass spectrometry abundance data can optionally undergo two preprocessing steps. The first step is a consolidation step: the goal is to to consolidate mass spectrometry observations in the data that are believed to belong to the same underlying compound. In other words, the instrumentation may have obtained multiple reads of mass spectrometry abundances that in actuality belong to the same compound - in which case we wish to attribute all of those observations to a single compound. The function name `binMS` derives from the fact that we use a binning procedure to consolidate the data.

The consolidation procedure is undergone as follows. Firstly, all observations must satisfy each of the following criterions or they are removed from consideration for consolidation (i.e. they are dropped from the data).

(i) Each observation must have its peak elution time occur during the specified interval.

(ii) Each observation must have a mass that falls within the specified interval.

(iii) Each observation must be detected in a charge state that falls within the specified interval.

Once the subset of observations satisfying the above criteria is obtained, a second step attempts to combine observations believed to belong to the same underlying compound. This procedure considers two observations that satisfy each of the following criterions to belong to the same compound.

(i) The absolute difference (in Daltons) of the mass-to-charge (m/z) values between the two observations is less the the specified value.

(ii) The absolute difference of the peak elution time between the two observations is less than the specified value.

(iii) The charge state must be the same for the two observations.

1.2 The `filterMS` function

The second optional preprocessing step for the mass spectrometry abundance data is a filtering step. The goal of the filtering step is to further reduce the data set to focus on only those compounds that could plausibly be contributing to the bioactivity area of interest. Furthermore, these criteria aim to filter out some of the noise detected in the dataset. By filtering the candidate set prior to statistical analysis, the ability of the analysis to effectively differentiate such compounds is greatly increased. The criteria for the downstream inclusion of a candidate compound are listed below.

(i) The m/z intensity maximum must fall inside the range of the bioactivity region of interest.

(ii) The ratio of the m/z intensity of a species in the areas bordering the region of interest and the species maximum intensity must be less than the specified value.

(iii) The right adjacent fraction to the fraction with maximum intensity for a given species must have a non-zero abundance.

(iv) At least 1 fraction in the region of interest must have intensity greater than the specified value.

(v) The compound charge state must be less than or equal to the specified value.

The region of interest is chosen to be a region where a high percent of bioactivity is observed in multiple sequential fractions; the assumption then is that there is some compound(s) eluting in those fractions that are associated with the observed bioactivity.

1.3 The `rankEN` function

Once the mass spectrometry abundance data has optionally undergone any preprocessing steps, a statistical procedure to search for putative bioactive peptides is performed. This step is performed by the `rankEN` function, which takes as inputs both the preprocessed mass spectrometry abundance data and the bioactivity data. The procedure works by specifying the level of the ℓ_2 penalty parameter in the elastic net penalty [2], and tracking the inclusion of the coefficients corresponding to compounds into the nonzero set along the elastic net path. An ordered list of candidate compounds is obtained by providing the order in which the coefficients corresponding to compounds entered the nonzero set.

1.4 The `msDat` function

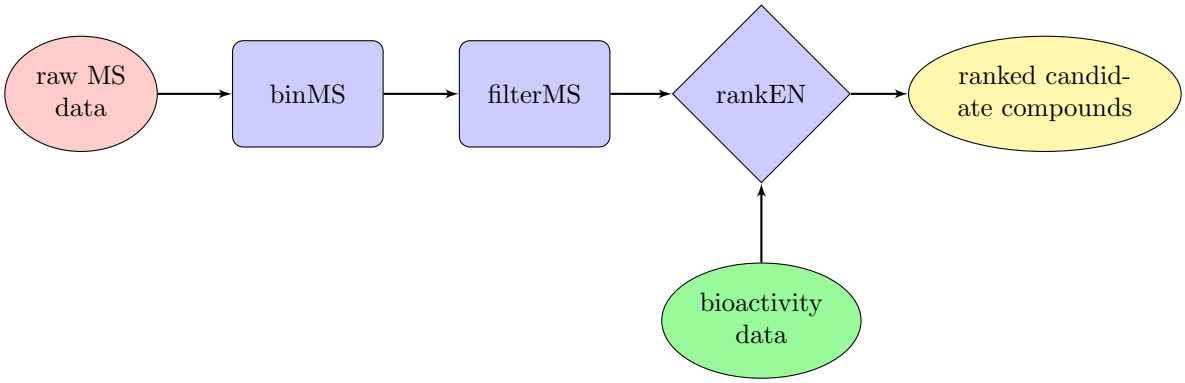
The functions `filterMS` and `rankEN` require objects of class `msDat` as the arguments providing the mass spectrometry abundance data; the `msDat` function takes mass spectrometry data as its input and creates an object of this class. Note however, that the `binMS` and `filterMS` functions return objects that inherit from the `msDat` class, and consequently you can provide an object created by `binMS` and `filterMS` anywhere that an `msDat` class object is required.

The raison d'être for an object of class `msDat` is to allow for a consistent interface to `filterMS` and `rankEN` whether the mass spectrometry data is obtained via a prior call to `binMS`, `filterMS`, or directly from user via a call to `msDat`.

2 An example case: performing the PepSAVI-MS pipeline

In the following section we will illustrate the usage of the core PepSAVIms functions by walking through the prototypical PepSAVI-MS pipeline. A flowchart for this prototypical PepSAVI-MS pipeline is shown below.

Figure 2: Data analysis flow chart for the data analysis performed in *The PepSAVI-MS pipeline*



2.1 Reading in the data

2.1.1 Reading in the mass spectrometry data

First we load some mass spectrometry data included with the package into memory.

```
# Load package into memory
library(PepSAVIms)

# The mass spectrometry data is provided as a data.frame
is.data.frame(mass_spec)

## [1] TRUE

# There are 30,799 mass-to-charge levels, and 38 variables
dim(mass_spec)

## [1] 30799    38

# The first four variables provide the m/z level, time of peak retention, mass,
# and charge state of each observation.  The remaining 34 variables are the mass
# spectrometric intensities for each compound across fractions 11 through 43, and fraction 47.
names(mass_spec)
```

```
##  [1] "m/z"                    "Retention time (min)"   "Mass"
##  [4] "Charge"                 "20150207_CLK_BAP_VO_11" "20150207_CLK_BAP_VO_12"
##  [7] "20150207_CLK_BAP_VO_13" "20150207_CLK_BAP_VO_14" "20150207_CLK_BAP_VO_15"
## [10] "20150207_CLK_BAP_VO_16" "20150207_CLK_BAP_VO_17" "20150207_CLK_BAP_VO_18"
## [13] "20150207_CLK_BAP_VO_19" "20150207_CLK_BAP_VO_20" "20150207_CLK_BAP_VO_21"
## [16] "20150207_CLK_BAP_VO_22" "20150207_CLK_BAP_VO_23" "20150207_CLK_BAP_VO_24"
## [19] "20150207_CLK_BAP_VO_25" "20150207_CLK_BAP_VO_26" "20150207_CLK_BAP_VO_27"
## [22] "20150207_CLK_BAP_VO_28" "20150207_CLK_BAP_VO_29" "20150207_CLK_BAP_VO_30"
## [25] "20150207_CLK_BAP_VO_31" "20150207_CLK_BAP_VO_32" "20150207_CLK_BAP_VO_33"
## [28] "20150207_CLK_BAP_VO_34" "20150207_CLK_BAP_VO_35" "20150207_CLK_BAP_VO_36"
## [31] "20150207_CLK_BAP_VO_37" "20150207_CLK_BAP_VO_38" "20150207_CLK_BAP_VO_39"
## [34] "20150207_CLK_BAP_VO_40" "20150207_CLK_BAP_VO_41" "20150207_CLK_BAP_VO_42"
## [37] "20150207_CLK_BAP_VO_43" "20150207_CLK_BAP_VO_47"
```

2.1.2 Reading in the bioactivity data

Next we load some bioactivity data included with the package into memory. The object `bioact` is a list containing bioactivity values (in replicates) for sweet violet against several bacterial and viral pathogens, as well as some cancer cell lines; we will arbitrarily choose the bioactivity data against the Gram-negative bacterium *E. coli* as an illustrative example, and create an object for this data from the element in the list.

```
# Load data into memory
data(bioact)

# bioact is a list with each element corresponding to bioactivity data
is.list(bioact)

## [1] TRUE

# Names of the elements in bioact
names(bioact)

## [1] "ec" "bc" "pc" "oc" "ab" "pa" "fg"

# Arbitrarily select one of the datasets for further examples
EC <- bioact$ec

# EC is provided as a data.frame
is.data.frame(EC)

## [1] TRUE

# EC contains data for 3 replicates and 44 fractions
dim(EC)

## [1]  3 44

# The names of the fractions for which bioactivity observations were obtained
names(EC)

##  [1] "20150207_CLK_BAP_VO_1"  "20150207_CLK_BAP_VO_2"  "20150207_CLK_BAP_VO_3"
##  [4] "20150207_CLK_BAP_VO_4"  "20150207_CLK_BAP_VO_5"  "20150207_CLK_BAP_VO_6"
##  [7] "20150207_CLK_BAP_VO_7"  "20150207_CLK_BAP_VO_8"  "20150207_CLK_BAP_VO_9"
## [10] "20150207_CLK_BAP_VO_10" "20150207_CLK_BAP_VO_11" "20150207_CLK_BAP_VO_12"
## [13] "20150207_CLK_BAP_VO_13" "20150207_CLK_BAP_VO_14" "20150207_CLK_BAP_VO_15"
## [16] "20150207_CLK_BAP_VO_16" "20150207_CLK_BAP_VO_17" "20150207_CLK_BAP_VO_18"
## [19] "20150207_CLK_BAP_VO_19" "20150207_CLK_BAP_VO_20" "20150207_CLK_BAP_VO_21"
```

```
## [22] "20150207_CLK_BAP_VO_22" "20150207_CLK_BAP_VO_23" "20150207_CLK_BAP_VO_24"
## [25] "20150207_CLK_BAP_VO_25" "20150207_CLK_BAP_VO_26" "20150207_CLK_BAP_VO_27"
## [28] "20150207_CLK_BAP_VO_28" "20150207_CLK_BAP_VO_29" "20150207_CLK_BAP_VO_30"
## [31] "20150207_CLK_BAP_VO_31" "20150207_CLK_BAP_VO_32" "20150207_CLK_BAP_VO_33"
## [34] "20150207_CLK_BAP_VO_34" "20150207_CLK_BAP_VO_35" "20150207_CLK_BAP_VO_36"
## [37] "20150207_CLK_BAP_VO_37" "20150207_CLK_BAP_VO_38" "20150207_CLK_BAP_VO_39"
## [40] "20150207_CLK_BAP_VO_40" "20150207_CLK_BAP_VO_41" "20150207_CLK_BAP_VO_42"
## [43] "20150207_CLK_BAP_VO_43" "20150207_CLK_BAP_VO_47"
```

2.2 Using `binMS`

Now that we have the mass spectrometry data loaded into memory, the first step in the PepSAVIms pipeline is to to consolidate mass spectrometry observations in the data that are believed to belong to the same underlying compound. This is performed via the `binMS` function.

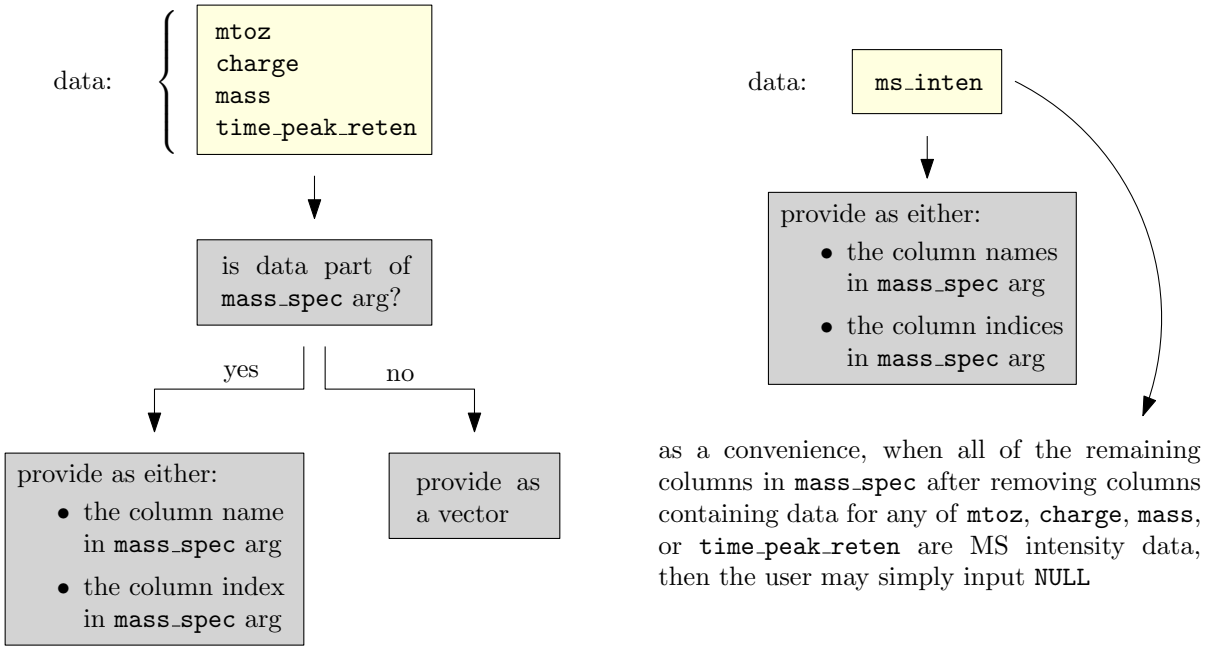
In order to perform its task, the consolidation routine needs to be provided with data for each mass-to-charge ratio and charge state, the retention time, and (optionally) the mass. These can be provided as data vectors, or can be given as columns in the mass spectrometry data. When provided as part of the mass spectrometry data, the variables can each be identified either by column index or by column name.

The mass spectrometry intensities are required to be included as part of the data provided as the argument to `mass_spec`. If all of the columns in `mass_spec` correspond either to mass spectrometry intensities or alternatively provide data for the m/z, charge, mass, or time of peak retention values, then we can specify the argument as `NULL` which indicates that all remaining values are to be included. In either case, we can always provide a vector either of column indices or of column names to specify which columns of `mass_spec` to include as mass spectrometry intensities.

As part of the procedure, we also need to specify acceptable values for the time of peak retention, an allowable mass range,and an allowable charge state range. These are provided as length-2 vectors specifying the lower and upper bounds of the acceptable ranges.

Finally, we need to specify the closeness of mass-to-charge values and times of peak retention for observations which we presume as having come from the same underlying compounds. This is done by specifying a cutoff value for which differences larger than these values will be considered as coming from two distinct mass-to-charge levels.

Figure 3: `binMS` data input flowchart



2.2.1 `binMS` specification via variable names

In this example we specify the data for the mass-to-charge, charge, mass, time of peak retention, and the intensities by specifying the names of the corresponding columns in `mass_spec`. Note that we do not need to provide the exact names of the columns, we only need to provide enough information to yield a unique match.

Further notice that we are able to provide `NULL` as the argument specifying the mass spectrometry intensities. Recall that the mass spectrometry data has columns for the m/z, charge, mass, and time of peak retention data, and that the remaining columns are intensities. Thus by specifying `NULL` this causes the function to include all of the remaining columns as intensity data after removing the m/z, charge, mass, and time of peak retention data.

```
# Perform consolidation using names
bin_out <- binMS(mass_spec = mass_spec,
                 mtoz = "m/z",
                 charge = "Charge",
                 mass = "Mass",
                 time_peak_reten = "Reten",
                 ms_inten = NULL,
                 time_range = c(14, 45),
                 mass_range = c(2000, 15000),
                 charge_range = c(2, 10),
                 mtoz_diff = 0.05,
                 time_diff = 60)
```

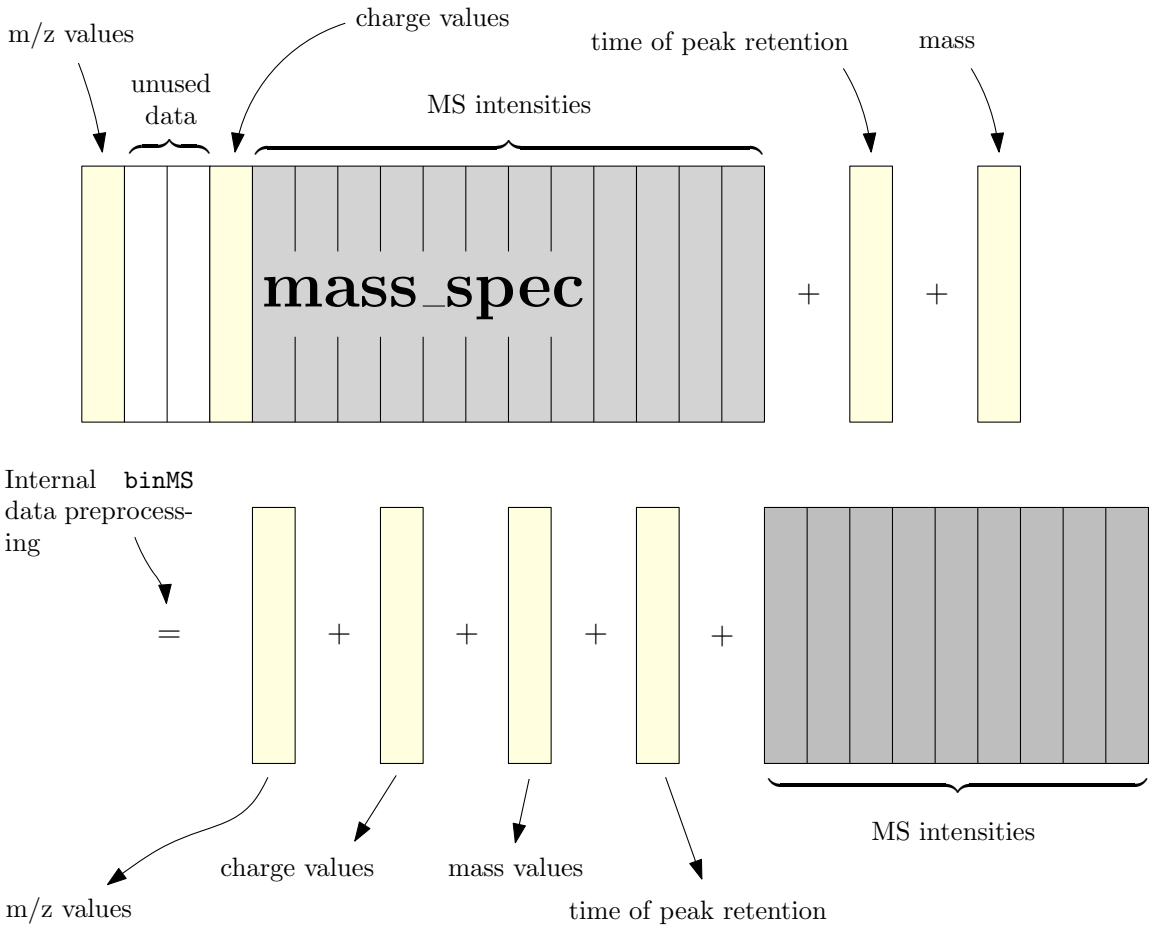
2.2.2 `binMS` via names, indices, and vectors

In the following example we explicitly provide the data for `mass` and `time_peak_retention` by passing data vectors as their arguments. Notice that now we have to specify the columns from `mass_spec` to use as intensity data, since after removing the columns containing the m/z and charge data, there are still non-intensity columns in `mass_spec` (specifically the columns containing the mass and time of peak retention data).

See Figure 4 for a visual schematic of the internal `binMS` processing of the data inputs for this following example.

```
# Make copies of some of the vectors in mass_spec to pass directly to function
mass_vals   <- mass_spec[, "Mass"]
time_vals   <- mass_spec[, "Retention time (min)"]

# Vector of names for the intensity columns.  We include the leading underscore
# so as to prevent any ambiguity between the fraction number and date.
inten_nm <- c(paste0("_", 11:43), "_47")

# Perform consolidation alternate input
bin_out_v2 <- binMS(mass_spec = mass_spec,
                    mtoz = "m/z",
                    charge = "Charge",
                    mass = mass_vals,
                    time_peak_reten = time_vals,
                    ms_inten = inten_nm,
                    time_range = c(14, 45),
                    mass_range = c(2000, 15000),
                    charge_range = c(2, 10),
                    mtoz_diff = 0.05,
                    time_diff = 60)

# We get the same results whether specifying data via column names or column
# indices
identical(bin_out_v2, bin_out)

## [1] TRUE
```

Figure 4: `binMS` data inputs corresponding to `bin_out_v2`



2.2.3 The print and summary function for `binMS`

The `binMS` class is equipped with a print and a summary function. We can see from the output that there were originally 30,799 mass-to-charge levels. After filtering by the inclusion criteria this reduced the data to 10,902 levels, and then consolidation of the data resulted in 6,258 levels.

```
# Print the size of the consolidated data
bin_out

## An object of class "binMS" with 6258 compounds and 34 fractions.
## Use summary to see more details regarding the consolidation process.
## Use extractMS to extract the consolidated mass spectrometry data.

# Show summary information describing the consolidation process
summary(bin_out)

##
## The inclusion criteria was specified as follows:
## ------------------------------------------------
##     time of peak retention:  between    14 and     45
##     mass:                    between 2,000 and 15,000
```

```
##     charge:                 between     2 and     10
##
## m/z levels were consolidated when each of the following criteria were met:
## ------------------------------------------------------------------------
##     m/z levels were no more than 0.05 units apart
##     the time peak retention occured no farther apart than 60 units
##     the charge states were the same
##
## The mass spectrometry data prior to binning had:
## ------------------------------------------------
##     30,799 m/z levels
##
## The number of remaining m/z levels after filtering by the inclusion criteria was:
## ------------------------------------------------------------------------------
##     time of peak retention:  26,387
##     mass:                    11,482
##     charge:                  19,250
##     satisfied all:           10,902
##
## After consolidating the m/z levels, there were:
## -----------------------------------------------
##     6,258 levels
```

2.3 Using `filterMS`

The next step in the PepSAVIms pipeline is to remove any potential candidate compounds with observed abundances for which it is scientifically unlikely that they might correspond to compound with an effect on the bioactivity area of interest. This step is performed by the `filterMS` function.

When using the `filterMS` function, the first task is to specify the region of interest and bordering region. This is done by providing an `msDat` object as an argument to `msObj`, and then:

(i) specifying a contiguous region of columns from the intensity matrix by either providing a vector either of column names or of column indices (`region` formal argument)

(ii) specifying the bordering region relative to the region of interest by providing either the value `"all"`, the value `"none"`, or a length-1 or length-2 vector provided the width of the left and right bordering regions (`border` formal argument)

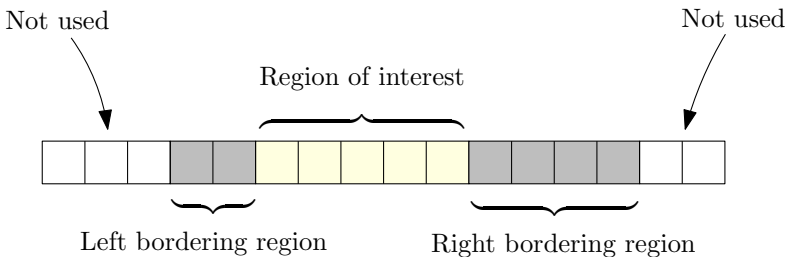


Figure 5: Region of interest and bordering regions

Once the region of interest and bordering region has been specified, the remaining variables `bord_ratio`, `min_inten`, and `max_chg` are each specified by selecting a single numeric value; see the function documentation for the precise definition of these variables.

2.3.1 `filterMS`: specifying the region of interest

In the following code snippet we provide examples of using `filterMS` with `region` specified either by using column names or by column indices.

```
# Invoke filterMS using column names to specify the region of interest
filter_out <- filterMS(msObj = bin_out,
                       region = paste0("VO_", 17:25),
                       border = "all",
                       bord_ratio = 0.01,
                       min_inten = 1000,
                       max_chg = 10)

# The column indices 7-15 correspond to fractions 17-25
colnames(filter_out)[7:15]

## [1] "20150207_CLK_BAP_VO_17" "20150207_CLK_BAP_VO_18" "20150207_CLK_BAP_VO_19"
## [4] "20150207_CLK_BAP_VO_20" "20150207_CLK_BAP_VO_21" "20150207_CLK_BAP_VO_22"
## [7] "20150207_CLK_BAP_VO_23" "20150207_CLK_BAP_VO_24" "20150207_CLK_BAP_VO_25"

# Invoke filterMS using indices to specify the region of interest
filter_out_v2 <- filterMS(msObj = bin_out,
                          region = 7:15,
                          border = "all",
                          bord_ratio = 0.01,
                          min_inten = 1000,
                          max_chg = 10)

# Confirm that the two objects are equivalent
identical(filter_out_v2, filter_out)

## [1] TRUE
```

2.3.2 `filterMS`: specifying bordering region

In the following code snippet we provide examples of using `filterMS` with `border` specified either by using a length-1 or length-2 numeric vector. Since in both cases the choices of `border` are large enough to encompass all of the fractions not included in the region of interest, these choices have the same effect as specifying `"all"`.

```
# Use one value to specify the width of both the left and the right bordering
# region
filter_out_v3 <- filterMS(msObj = bin_out,
                          region = paste0("VO_", 17:25),
                          border = 100,
                          bord_ratio = 0.01,
                          min_inten = 1000,
                          max_chg = 10)

# Use two values to specify the left width and right width of the bordering
# region
filter_out_v4 <- filterMS(msObj = bin_out,
                          region = paste0("VO_", 17:25),
                          border = c(150, 200),
                          bord_ratio = 0.01,
                          min_inten = 1000,
                          max_chg = 10)
```

```
# We get the same result be specifying the left and right bordering regions as
# having widths 100 as by choosing "all"
identical(filter_out_v3$msDatObj, filter_out$msDatObj)

## [1] TRUE

# We get the same result be specifying the left and right bordering regions as
# having widths 150 and 200 as by choosing "all"
identical(filter_out_v4$msDatObj, filter_out$msDatObj)

## [1] TRUE
```

2.3.3 The print and summary function for `filterMS`

The `filterMS` class is equipped with a print and a summary function. We see that the number of candidate compounds is reduced from 6,258 compounds to 225. Note that when the number of fractions in the region of interest or the bordering regions is large, then the summary function omits printing the fractions so as to prevent the output from becoming overly lengthy - in this case the faction names for the bordering region is omitted.

```
# Print the size of the filtered data
filter_out

## An object of class "filterMS" with 225 compounds and 34 fractions.
## Use summary to see more details regarding the filtering process.
## Use extractMS to extract the filtered mass spectrometry data

# Show summary information describing the filtering process
summary(filter_out)

##
## The region of interest was specified as (9 fractions):
## ------------------------------------------------------
##     20150207_CLK_BAP_VO_17
##     20150207_CLK_BAP_VO_18
##     20150207_CLK_BAP_VO_19
##     20150207_CLK_BAP_VO_20
##     20150207_CLK_BAP_VO_21
##     20150207_CLK_BAP_VO_22
##     20150207_CLK_BAP_VO_23
##     20150207_CLK_BAP_VO_24
##     20150207_CLK_BAP_VO_25
##
## The bordering regions were specified as "all"
## ---------------------------------------------
##     * fraction names omitted for brevity *
##
## The filtering criteria was specified as:
## ----------------------------------------
##     minimum intensity:      1,000
##     maximum charge:            10
##     bordering region ratio:     0.01
##
## The mass spectrometry data prior to filtering had:
## --------------------------------------------------
##     6,258 compounds
##        34 fractions
##
```

```
## Individually, each criterion reduced the 6,258 m/z levels to the following number:
## ------------------------------------------------------------------------------
##     criterion 1:  3,428    (fraction with max. abundance is in region of interest)
##     criterion 2:  1,080    (fractions in bordering region have < 1% of max. abundance)
##     criterion 3:  2,818    (nonzero abundance in right adjacent fraction to max.)
##     criterion 4:  4,012    (at least 1 intensity > 1,000 in region of interest)
##     criterion 5:  6,258    (must have charge <= 10)
##
## The total number of candidate compounds was reduced to:
## ---------------------------------------------------------
##     225
```

2.4 Using `rankEN`

Once the mass spectrometry abundance data has optionally undergone any preprocessing steps, a statistical procedure to search for candidate compounds for reduction of bioactivity levels is performed. This step is performed by the `rankEN` function, and takes as inputs both the preprocessed mass spectrometry abundance data and the bioactivity levels data.

2.4.1 `rankEN`: specifying the region of interest for mass spectrometry and bioactivity data

The first task in invoking the `rankEN` procedure is to specify the the region of interest for mass spectrometry and bioactivity data. This can be done by specifying the appropriate column names or column indices in the respective data. So the argument for `region_ms` should specify the region of interest for the mass spectrometry data by providing the appropriate column names or column indices with respect to the argument provided for `msObj`. Similarly, the argument for `region_bio` should be with respect to the argument for `bioact`. It is worth clarifying that it should be the same region of interest for the intensity and bioactivity data (i.e. the region should correspond to the same fractions for each); it just might be the case that the column names or indices may differ.

Once the mass spectrometry and bioactivity data has been provided and the region of interest for each has been specified, it remains to specify

- the quadratic penalty paramter for the elastic net penalty
- a switch specifying whether the function should retain only compounds that are positively correlated the bioactivity
- the maximum number of candidate compounds to retain

See the function documentation for more detail.

```
# Perform the candidate ranking procedure with fractions 21-24 as the region of
# interest
rank_out <- rankEN(msObj = filter_out,
                   bioact = EC,
                   region_ms = paste0("_", 21:24),
                   region_bio = paste0("_", 21:24),
                   lambda = 0.001,
                   pos_only = TRUE,
                   ncomp = NULL)
```

2.4.2 The summary function for `rankEN`

The `rankEN` class is equipped with a print and summary function. The summary function provides a list of the candidate compounds obtained by the procedure and their correlation with mean bioactivity levels across replicates.

```r
# Prints the dimensions of the data
rank_out
```

```
##
## An object of class "rankEN".
## Use summary to print a list of the compounds coefs along the elastic net path.
## Use extract_candidates to extract the compound info as a data.frame.
##
## Data dimensions:
## ----------------
##     region of interest:       4
##     candidate compounds:    225
##     bioactivity replicates:   3
```

```r
# Shows the first 10 candidate compounds obtained by the procedure
summary(rank_out, 10)
```

```
##
## Data dimensions (some compounds were removed):
## ----------------------------------------------
##     region of interest:        4
##     candidate compounds:     225
##     bioactivity replicates:    3
##
## Compounds removed for being constant:
## -------------------------------------
##     Mass-to-charge    Charge
##     --------------    ------
##       838.4148243     3
##     1,079.154755      5
##     1,079.553245      4
##     1,139.2007925     3
##     1,142.5517        3
##     1,199.648221      2
##     1,279.7839235     5
##     1,415.2402355     2
##     1,591.255737      2
##
## Fractions included in region of interest:
## -----------------------------------------
##     Mass spec.                Bioactivity
##     ----------                -----------
##     20150207_CLK_BAP_VO_21    20150207_CLK_BAP_VO_21
##     20150207_CLK_BAP_VO_22    20150207_CLK_BAP_VO_22
##     20150207_CLK_BAP_VO_23    20150207_CLK_BAP_VO_23
##     20150207_CLK_BAP_VO_24    20150207_CLK_BAP_VO_24
##
## Parameter arguments provided to rankEN:
## ---------------------------------------
##     Quadratic penalty parameter:          0.001
##     Consider only positive correlations:  yes
##     Max number of candidate compounds:    all
##
## Compounds in order of entrance (first 10 compounds, earliest at top):
## ----------------------------------------------------------------------
##     Mass-to-charge       Charge    Correlation
##     --------------       ------    -----------
```

```
##     1,054.48586583333    3         0.9998
##     1,580.7535535        2         0.9996
##     1,580.252571         2         0.9996
##     1,278.212122         5         0.9995
##     1,093.840501         3         0.9963
##     1,056.49074633333    3         0.9967
##     1,383.8082405        7         0.9991
##     1,592.252869         4         0.9991
##     1,276.8032336        5         0.9896
##     1,258.203348         5         0.9977
```

2.4.3 Accessing the ranked candidate compounds

The m/z and charge values of the ranked candidate compounds as well as their correlations with respect to the average bioactivity levels for the region of interest can be extracted and returned as a `data.frame` via the `extract_ranked` function.

```
# Extract the ranked candidates
ranked_candidates <- extract_ranked(rank_out)

# Return object is a data.frame
is.data.frame(ranked_candidates)

## [1] TRUE

# Print first few candidates; should be the same as from the summary function
head(ranked_candidates)

##       mtoz charge  comp_cor
## 1 1054.486      3 0.9998253
## 2 1580.754      2 0.9996284
## 3 1580.253      2 0.9995584
## 4 1278.212      5 0.9995421
## 5 1093.841      3 0.9962525
## 6 1056.491      3 0.9967004
```

3 Data access and manipulation tools

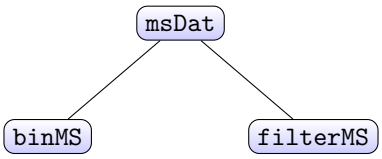
In this section we take a deeper look into the objects created by the functions in PepSAVIms and consider how to further access and manipulate the data throughout the process.

3.1 The mass spectrometry data class hierarchy

The core data structure in the PepSAVIms package is the `msDat` *class*; it is the class of the object returned by the `msDat` *function*. This data structure is used to maintain mass spectrometry data. See the `msDat` function documentation for further details regarding the `msDat` class internal structure.

The importance of the `msDat` class lies in the fact that the `binMS` and `filterMS` *classes* derive from it; these classes are the objects that are returned from the `binMS` and `filterMS` *functions*. The `binMS` and `filterMS` classes are essentially objects that decorate the `msDat` class; each of these data structures includes additional information used by the class's respective summary function to describe the data processing procedure. See the respective function documentation for more detail regarding the internal structure of the `binMS` and `filterMS` classes.

Figure 6: The PepSAVIms class hierarchy



3.2 The `extractMS` function

The `extractMS` function is a convenience function taking an object inheriting from the `msDat` class and returning the encapsulated mass spectrometry data as either (as specified by the user):

i) a matrix

ii) an `msDat` object (strictly an `msDat` object, i.e. not a subclass)

3.2.1 Extracting a `matrix` object

The user may find it convenient to view and / or manipulate the mass spectrometry data encapsulated in an `msDat` object as a data matrix. This data may always be refactored again as an `msDat` object via the `msDat` function. Converting an `msDat` object to a matrix is done by specifying `type` as `"matrix"`.

The matrix object returned by the `msDat` function has a form where the first two columns provide the mass-to-charge and charge values of the data respectively, and the remaining columns provide the intensity data across the fractions.

One important situation where the user may wish to refactor the mass spectrometry data into the form of a matrix is as an intermediate step if they wish to convert the data to for example a comma-separated values file or native spreadsheet data format.

```
# Refactor the data as a matrix
filter_matr <- extractMS(msObj = filter_out, type = "matrix")

# Return object is a matrix
is.matrix(filter_matr)

## [1] TRUE

# The data has two extra columns, one each for the m/z and charge information
dim(filter_matr)

## [1] 225  36

# Compare to the result of calling dim on the original msDat object
dim(filter_out)

## [1] 225  34

# Print the first few rows and columns of the newly formed matrix.  The row
# names of the matrix are the concatonation of the mass-to-charge ratio and
# charge state, separated by a /.
filter_matr[1:5, 1:4]

##                mtoz charge 20150207_CLK_BAP_VO_11 20150207_CLK_BAP_VO_12
##  412.5575/6 412.5575      6                     53                      0
##  425.8807/9 425.8807      9                      0                      0
##  484.8031/7 484.8031      7                      0                      0
##  581.9921/5 581.9921      5                      0                      0
##  716.8877/7 716.8877      7                      0                      0
```

Once the data has been refactored in matrix form, any of the usual R data export tools can be used.

```
# Save the data as a csv file.  Probably don't want to keep the row names as that
# information is contained in the first two columns of the data.
write.csv(filter_matr, file = "filtered_mass_spec.csv", row.names = FALSE)
```

3.2.2 Extracting an `msDat` object

An alternative to refactoring a `binMS` or `filterMS` object as a `matrix` is to extract the internal `msDat` object. This is done by specifying `type` as `"msDat"`. A potential advantage of keeping the data as an `msDat` object is that it may prevent later converting the data back from a matrix into an `msDat` object.

The `msDat` class is equipped with fundamental operations common to typical matrix classes such as data printing and matrix subsetting (described in section 3.4). The main difference between the extracted `msDat` object and its encapsulating `binMS` or `filterMS` object is that the print and summary functions for the `msDat` emulates the print and summary functions for a matrix of intensity values rather than describing the consolidation or filtering process.

```
# Extract the encapsulated msDat object
filter_msDat <- extractMS(filter_out, "msDat")

# For a subclass of msDat the extractMS function has the effect of performing
# the following command
filter_msDat_v2 <- filter_out$msDatObj

# extractMS is the same as copying the msDatObj element for a subclass of msDat
identical(filter_msDat_v2, filter_msDat)

## [1] TRUE

# Calling extractMS on an object that is strictly of class msDat is effectively
# a noop
filter_msDat_v3 <- extractMS(filter_msDat, "msDat")

# extractMS on a strictly msDat object returns the original object
identical(filter_msDat_v3, filter_msDat)

## [1] TRUE

# Printing the extracted msDat object prints the intensity matrix (as opposed to
# the print function for binMS or filterMS objects.  Also compare this to the
# extracted matrix in the previous section: in this form the mass-to-charge and
# charge data is not exposed to the user.
filter_msDat[1:5, 1:2]

##             20150207_CLK_BAP_VO_11 20150207_CLK_BAP_VO_12
##  412.5575/6                     53                      0
##  425.8807/9                      0                      0
##  484.8031/7                      0                      0
##  581.9921/5                      0                      0
##  716.8877/7                      0                      0
```

3.3 The `msDat` function

As might be expected, the `msDat` *function* takes mass spectrometry data as input and returns an `msDat` *object*. In one sense the `msDat` function can be though of as the inverse of the `extractMS` function with `type` specified as

"matrix"; while in this form extractMS turns an msDat object into a matrix, the msDat function turns a matrix into an msDat object.

In general, the msDat function is used to create an msDat object for use as input for either the filterMS or the rankEN function. This need may occur when the researcher wishes to (i) enter the PepSAVI-MS pipeline without executing either or both of the binMS or filterMS functions or (ii) wants to do some addition processing between steps of the pipeline using the raw data.

The forms in which the data can be input is similar to that as for the binMS function, so we do not go into great deal here. In fact the internal routines used to process the arguments are the same for both functions.

```
# Construct an msDat object from object created by a call to extractMS
filter_out_v5 <- msDat(mass_spec = filter_matr,
                       mtoz = "mtoz",
                       charge = "charge",
                       ms_inten = NULL)

# Confirm that reconstructed msDat object is equal.  Need to ignore attributes
# when testing for equality b/c row names are not retained.
all.equal(filter_out_v5, filter_out$msDatObj, check.attributes=FALSE)

## [1] TRUE
```

3.4 The msDat class interface

The msDat class and its subclasses binMS and filterMS are equipped with some basic class methods to support fundamental data operations. The basic operations that are supported are the following:

- dim, nrow, ncol (in terms of the dimensions of the intensity data)
- dimnames, colnames, and row.names read / write
- extract or replace via the [ ] operator
- print

We've used many of these functions throughout the rest of this document without explanation, but now let us provide some concrete examples.

```
# Check the dimension; can also use nrow, ncol
dim(filter_msDat)

## [1] 225  34

# Print the first few rows and columns
filter_msDat[1:5, 1:3]

##             20150207_CLK_BAP_VO_11 20150207_CLK_BAP_VO_12 20150207_CLK_BAP_VO_13
##  412.5575/6                     53                      0                      0
##  425.8807/9                      0                      0                      0
##  484.8031/7                      0                      0                      0
##  581.9921/5                      0                      0                      0
##  716.8877/7                      0                      0                      0

# Let's change the fraction names to something more concise
colnames(filter_msDat) <- c(paste0("frac", 11:43), "frac47")

# Print the first few rows and columns with the new fraction names
filter_msDat[1:5, 1:10]
```

```
##             frac11 frac12 frac13 frac14 frac15 frac16 frac17   frac18 frac19 frac20
##  412.5575/6     53      0      0      0      0      0      0  87.2347    116      0
##  425.8807/9      0      0      0      0      0      0      0   0.0000      0      0
##  484.8031/7      0      0      0      0      0      0      0 207.0000      0      0
##  581.9921/5      0      0      0      0      0      0      0   0.0000      0    826
##  716.8877/7      0      0      0      0      0      0      0   0.0000      0   1038
```

```
# Suppose there are some m/z levels that we wish to remove
filter_msDat <- filter_msDat[-c(2, 4), ]

# Print the first few rows and columns after removing rows 2 and 4
filter_msDat[1:5, 1:10]
```

```
##              frac11 frac12 frac13 frac14 frac15 frac16 frac17   frac18 frac19 frac20
##  412.5575/6      53      0      0      0      0      0      0  87.2347    116      0
##  484.8031/7       0      0      0      0      0      0      0 207.0000      0      0
##  716.8877/7       0      0      0      0      0      0      0   0.0000      0   1038
##  740.7493/3       0      0      0      0      0      0      0 170.0000      0   5240
##  777.0136/10      0      0      0      0      0      0      0   0.0000      0   1512
```

```
# Suppose that there was an instrumentation error and that we need to change
# some values
filter_msDat[1, paste0("frac", 12:17)] <- c(55, 57, 62, 66, 71, 79)

# Print the first few rows and columns after changing some of the values in
# the first row
filter_msDat[1:5, 1:10]
```

```
##              frac11 frac12 frac13 frac14 frac15 frac16 frac17   frac18 frac19 frac20
##  412.5575/6      53     55     57     62     66     71     79  87.2347    116      0
##  484.8031/7       0      0      0      0      0      0      0 207.0000      0      0
##  716.8877/7       0      0      0      0      0      0      0   0.0000      0   1038
##  740.7493/3       0      0      0      0      0      0      0 170.0000      0   5240
##  777.0136/10      0      0      0      0      0      0      0   0.0000      0   1512
```

References

[1] Kirkpatrick, C.L., Broberg, C.A., McCool, E.N., Lee, W.J., Chao, A., McConnell, E.W., Hebert, M., Fleeman, R., Adams, J. and Jamil, A. (2017). The "PepSAVI-MS" Pipeline for Natural Product Bioactive Peptide Discovery. Analytical chemistry, 89(2), 1194-1201.

[2] Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.