Processing samples from GC-TOF-MS

Cristian Díaz-Muñoz & Hannes Decadt

v3.2.

Introduction

This is a tutorial meant to guide you through a semi-automatic pipeline to process the samples analyzed through headspace/solid-phase microextraction coupled to gas chromatography with time-of-flight mass spectrometry (HS/SPME-GC-TOF-MS). Currently, we use this technique as an untargeted screening tool, in other words, to scan which volatile organic compounds are in the samples. This is, hence, a semi-quantitative approach, as it serves to compare the samples between each other. To do that, we need to include an internal standard (IS) with the sample at the time of the analysis (typically Toluene-D8). We will ultimately use the peak areas of the compounds divided by the IS to generate a principal component analysis (PCA), which will illustrate the dissimilarity between our samples.

Pre-requisites

Before running the script you need to fill this checklist:

1) Output the processed chromatograms using the output format "Sample summary report", which should output an .xls file with only one sheet in it.
2) Do not output the MQ samples. If you did so, remove them before running the script.
3) The .xls files need to be in a separate folder inside your working directory. Name that folder "SSR" (from Sample Summary Report).
4) Your samples need to be named as follows: sample1_a.xls, sample1_b.xls, sample1_c.xls, sample2_a.xls, sample2_b.xls, sample2_c.xls, ... If your samples are not named like this and need to be renamed I show an example below of how to change them in R.
5) Create another folder named "Output reports".

Usage

The script presented in this guide is based on ScriptTof.R from Hannes Decadt and it is updated by Cristian Díaz-Muñoz –> ScriptTof_v3.R

The script takes the "sample summary report" output files from the GC-TOF-MS and process them doing:

1) Reshape the dataframe and get the best hit for each RT.
2) Flag all duplicated compounds and merge all files together.

3) Eliminate those compounds present in only 1 out of 3 samples.

4) Calculate the average of the triplicates.

Let's get into the matter. We will use an example with samples from Kombucha:

Import all needed libraries to run the script.

```
library(readxl) # To open excel files
library(tidyverse) # Type of R syntax, to make code easier
library(reshape2) # To transform dataframes
```

Set the working directory.

```
setwd("C:/Users/mmorenoc/Vrije Universiteit Brussel/Cristian DIAZ MUNOZ - Mery/TOF")
samples_wd <- "C:/Users/mmorenoc/Vrije Universiteit Brussel/Cristian DIAZ MUNOZ - Mery/TOF/SSR/"
```

List all files in the folder (all your samples).

```
files <- list.files(path = samples_wd, pattern=".xls")
# This option if your samples already have the correct names:
names <- as.vector(sapply(strsplit(basename(files), ".xls"), `[`, 1))
# This option if your samples do not contain any identifier in their filename:
names_2 <- c(rep(c("a", "b","c"),27))
names_3 <- c(rep("Captain_1", 3),rep("Captain_2", 3),rep("Captain_3", 3),
             rep("Karma_1", 3),rep("Karma_2", 3),rep("Karma_3", 3),
             rep("Thylbert_1", 3),rep("Thylbert_2", 3),rep("Thylbert_3", 3),
             rep("Rish_1", 3),rep("Rish_2", 3),rep("Rish_3", 3),
             rep("Gremline_1", 3), rep("Gremline_2", 3), rep("Gremline_3", 3),
             rep("Yaya_1", 3),rep("Yaya_2", 3),rep("Yaya_3", 3),
             rep("Smile_1", 3),rep("Smile_2", 3),rep("Smile_3", 3),
             rep("Yuguen_1", 3),rep("Yuguen_2", 3),rep("Yuguen_3", 3),
             rep("Bekombucha_1", 3),rep("Bekombucha_2", 3),rep("Bekombucha_3", 3))
names_f <- paste(names_3,names_2, sep = "_")
```

For the latter option, we just created a vector (names_2) with a repetition of "a", "b", and "c" (for the triplicate) a number of times equal to the number of samples. Then we create a second vector (names_3) which contains the sample names and we repeat every name three times. Finally, we just concatenate both vectors (names_f).

1) Reshape the dataframe and get the best hit for each RT

```
for (i in 1:length(files)) {
  # Import all files. 32 first rows are always empty, so can be skipped.
  df <- read_excel(path = paste("SSR/",files[i], sep = ""), skip=32)
  # Select the RT rows.
  rt <- as.numeric(df[,12]=="DBC TIC")
  # In which position they are.
  index <- which(rt %in% 1)
  # Create a data frame with all information needed.
  df1 <- data.frame(Compound = as.data.frame(df[index + 1, 1]),
                    CAS = as.data.frame(df[index + 1, 17]),
```

```
                    RT = as.data.frame(df[index, 1]),
                    MF = as.data.frame(df[index + 1, 19]),
                    area = as.data.frame(df[index, 22])
  )
  # Modify the column names.
  colnames(df1) <- c("Compound", "CAS", "RT", "MF", "Area")
  # Write the table in tsv format (easy to work with in R and Excel)
  write.table(file = paste("SSR/",names_f[i],".txt", sep = ""),
              x = df1, sep = "\t", dec = ".", quote = FALSE, row.names = FALSE)
}
```

Check that the .txt files generated are okay and that there is not any "NA.txt".

2) Flag all duplicated compounds and merge all files together

First, we need to list all files in the folder (all txt files in this case, which we just generated above). Is very important to sort the names_f, as you will import the new files in alphabetical order.

```
files2 <- list.files(path = samples_wd, pattern=".txt")
names_f2 <- sort(names_f)
```

Create an empty data frame.

```
df_all <- data.frame(matrix(ncol = 7, nrow = 0))
colnames(df_all) <- c("Compound", "CAS", "RT", "MF", "Area", "Sample","Duplicated")
```

Run the next for loop to proceed to the flagging of duplicated compounds.

```
for (i in 1:length(files2)) {
  # Import all the files.
  df2 <- read.delim(file = paste("SSR/",files2[i], sep = ""))
  # Put the sample name.
  df2$Sample <- rep(names_f2[i], nrow(df2))
  # Some compounds can be found multiple times: list all of them up.
  df2$Duplicated <- duplicated(df2$Compound)
  # In which position they are.
  index2 <- which(duplicated(df2$Compound))
  # Add an '*' to every compound that is found twice (two *'s if it is found 3 times, etc.).
  while (length(index2)>0){
    for (k in index2){
      df2$Compound[k] <- str_c(df2$Compound[k],"*")
    }
    index2 <- which(duplicated(df2$Compound))
  }
  df_all <- rbind(df_all,df2)
}
```

Write the table in tsv format (easy to work with in R and Excel).

```
write.table(file = "Output reports/all_samples.txt",
            x = df_all, sep = "\t", dec = ".", quote = FALSE, row.names = FALSE)
```

A good **check** to ensure that everything is going alright is to look at a specific compound from a specific sample and check if the RT and Area are the same in "all_samples.txt" and the individual sample from the previous step.

We are also going to perform an **ESSENTIAL CHECK**. We need to know if all samples have been processed correctly in the software of the TOF. To do a fast check of that, we are going to see if Toluene-D8 (or the IS you used, change accordinglt then) is present in all samples. If it is not present, it possibly means that the peak identification and processing of that sample is wrong. So you should go back to the TOF and reprocess those samples. Then, run the script from the beginning.

```
check_list <- df_all$Sample[df_all$Compound == "Toluene-D8"]
print(paste("There are ", length(files2) - length(check_list), " files with no Toluene-D8"))
```

3) Eliminate those compounds present in only 1 out of 3 samples

Add a column with the triplicate info (change according to the name of the file). We first create a copy of the previous df, but as a new variable (df_tri). Then, we divide the column sample by the underscore ("_"), select the third field, and output it as a new column called Rep. Finally, we eliminate the third field from the column Sample.

```
df_tri <- df_all
df_tri$Rep <- sapply(strsplit(basename(as.character(df_tri$Sample)), "_"), `[`, 3)
df_tri$Sample <- paste(sapply(strsplit(basename(as.character(df_tri$Sample)), "_"), `[`, 1),
                       sapply(strsplit(basename(as.character(df_tri$Sample)), "_"), `[`, 2),
                       sep = " ")
```

Calculate the standard deviation for each compound in the triplicate. The aim behind this is that if there is one compound that is only present in 1 out of 3 samples, the SD will be NA. We will then filter them out using this.

```
df_tri <- df_tri %>%
  group_by(Compound, Sample) %>%
  mutate(Area_SD = sd(Area))
```

Eliminate all the compounds with non-existing SD (present in only 1 sample).

```
df_cur <- subset(df_tri, df_tri$Area_SD > 0)
```

We can now count how many duplicates there are still in the samples. In this case, before and after removing the compounds present in only 1 sample.

```
length(df_all$Compound[df_all$Duplicated == TRUE])
length(df_cur$Compound[df_cur$Duplicated == TRUE])
```

Write the table in tsv format (easy to work with in R and Excel).

```
write.table(file = "Output reports/all.samples.curated.txt",
            x = df_cur, sep = "\t", dec = ".", quote = FALSE, row.names = FALSE)
```

4) Calculate the average of the triplicates

Calculate the mean for every numeric column and remove the replicates.

```
df_av <- df_cur %>%
  group_by(Compound, Sample) %>%
  mutate(RT_SD = sd(RT)) %>%
  summarize_if(is.numeric, mean, na.rm = TRUE)
```

Write the table in tsv format (easy to work with in R and Excel)

```
write.table(file = "Output reports/all.samples.curated.averaged.txt",
            x = df_av, sep = "\t", dec = ".", quote = FALSE, row.names = FALSE)
```

Post-processing check points

You can now use the data frames generated to inspect the data. An easy way would be to open them in different sheets of an Excel file. You should first look at the RT_SD. If this number is too big, you can be suspicious about the peak and you can maybe flag it to check it manually. Of course, all duplicated compounds (*) should be checked to determine which is the correct one for that compound and to which compound truly correspond the other peaks. Finally, you should also check that you don't have any compound that do not vary over all samples, as this is first not very reliable and second will cause problems in later stages.

Once you have manually corrected all these inaccuracies we can proceed to the next step of the workflow, which is to perform a PCA.

Principal Component Analysis