Optimising identification and quantitation by combining data using the *synapter* package

Laurent Gatto*, Pavel V. Shliaha and Sebastian Gibb

October 17, 2016

Abstract

This vignette describes the functionality implemented in the *synapter* package. It allows to re-analyse label-free quantitative proteomics data obtained on a Synapt G2 instrument to optimise quantitation and identification. Several combination strategies are possible and described. Typically, a user can combine identification-optimised data (HDMS^E data using ion mobility separation) and quantitation-optimised data (MS^E data). Additionally, a method to combine several data files into a *master* set while controlling the false discovery rate, is presented.

Keywords: Mass Spectrometry (MS), proteomics, bioinformatics, IMS, ion mobility separation ion mobility, Synapt, label free, data independent acquisition.

*lg390@cam.ac.uk

Contents

1 Introduction

1.1 Background

The main functionality of *synapter* is to combine proteomics data acquired under different mass spectrometry settings or with different samples to (i) optimise the respective qualities of the two data sets or (ii) increase the number of identifications, thereby decreasing missing values. Besides synapter offers other functionality inaccessible in the default pipeline, like peptide FDR estimation and filtering on peptide match type and peptide uniqueness.

The example that motivated the development of this package was to combine data obtained on a Synapt G2 instrument:

1. HDMS^E data, acquired with additional peptide separation using an ion mobility cell, thus leading to better (both in number and in quality) identification and
2. standard MS^E data (acquired without ion mobility separation), providing better data quantitation.

The former is data is called *identification peptides* and the latter *quantitation peptides*, irrespective of the acquisition mode (HDMS^E or MS^E). This HDMS^E/MS^E design is used in this document to illustrate the *synapter* package.

However, although HDMS^E mode possesses superior identification and MS^E mode superior quantitation capabilities and transferring identifications from HDMS^E to MS^E is a priori the most efficient setup, identifications can be transferred between any runs, independently of the acquisition mode. This allows to reduce the number of missing values, one of the primary limitation of label-free proteomics. Thus users will benefit from *synapter*'s functionality even if they run their instruments in a single mode (HDMS^E or MS^E only).

However, as will be shown in section **??**, transferring identifications from multiple runs to each other increases analysis time and peptide FDR within the analysis. *synapter* allows to minimise these effects to acceptable degree by choosing runs to transfer identifications from and merging them in the *master* HDMS^E file.

This data processing methodology is described in section **??** and the analysis pipeline is described in section **??**.

To maximise the benefit of combining better identification and quantitation data, it is also possible to combine several, previously merged identification data files into one *master* set. This functionality is described in section **??**.

Finally, section **??** illustrates a complete pipeline including *synapter* and *MSnbase* [**?**] packages to perform protein label-free quantitation: how to combine multiple *synapter* results to represent the complete experimental design under study and further explore the data, normalise it and perform robust statistical data analysis inside the *R*environment.

The rationale underlying *synapter*'s functionality are described in [**?**] and [**?**]. The first reference describes the benefits of ion mobility separation on identification and the effects on quantitation, that led to the development of *synapter*, which in described and demonstrated in [**?**].

synapter is written for *R* [**?**], an open source, cross platform, freely available statistical computing environment and programming language[1]. Functionality available in the *R*environment can be extended though the usage of packages. Thousands of developers have contributed packages that are distributed via the Comprehensive R Archive Network (CRAN) or through specific initiatives like the Bioconductor[2] project [**?**], focusing on the analysis and comprehension of high-throughput biological data.

synapter is such an *R*package dedicated to the analysis of label-free proteomics data. To obtain detailed information about any function in the package, it is possible to access it's documentation by preceding it's name with a question mark at the command line prompt. For example, to obtain information about the *synapter* package, one would type `?synapter`.

1.2 Installation

synapter is available through the Bioconductor project. Details about the package and the installation procedure can be found on its page[3]. Briefly, installation of the package and all its dependencies should be done using the dedicated Bioconductor infrastructure as shown below:

```
> source("http://bioconductor.org/biocLite.R")
> ## or, if you have already used the above before
> library("BiocInstaller")
> ## and to install the package
> biocLite("synapter")
```

After installation, *synapter* will have to be explicitly loaded with

```
> library(synapter)
```

so that all the package's functionality is available to the user.

1.3 Getting help

Question should be asked on the Bioconductor support forum[4]. Bugs and feature requests can also be reported on the github tracker [5].

synapter is an open source initiative and contributions, whether new code, documentation bug fixes and new use cases are much appreciated. The official source code is available on the Bioconductor svn server[6]. A testing version and easily fork-able source tree is available on github[7].

[1] http://www.r-project.org/
[2] http://www.bioconductor.org/
[3] http://bioconductor.org/packages/devel/bioc/html/synapter.html
[4] https://support.bioconductor.org
[5] https://github.com/lgatto/synapter/issues
[6] http://bioconductor.org/developers/source-control/
[7] https://github.com/lgatto/synapter/

2 Data analysis using synapter

2.1 Preparing the input data

Preparation of the data for *synapter* requires the `.raw` data first to be processed with Waters' ProteinLynx Global Serve (PLGS) software. The PLGS result is then exported as `csv` spreadsheet files in user specified folders. These `csv` files can then be used as input for *synapter*.

We also highly recommend users to acquaint themselves with the PLGS search algorithm for data independent acquisitions [**?**].

First the user has to specify the output folders for files to be used in synapter analysis as demonstrated in figure **??**. After the folders are specified ignore the message that appears requiring restarting PLGS.

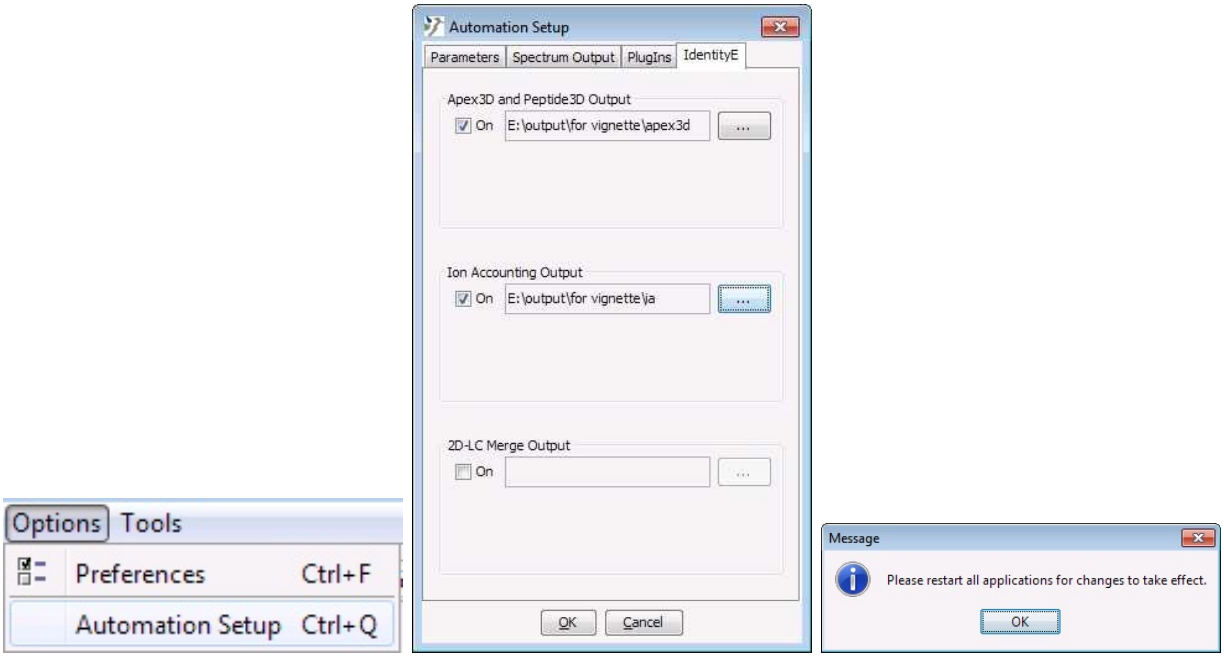


Figure 1: Specifying PLGS output folders. The last message can be ignored.

At the first stage PLGS performs noise reduction and centroiding, based on user specified preferences called *processing parameters*. These preferences determine thresholds in intensity for discriminating between noise peaks and peptide and fragment ion peaks in high and low energy functions of an acquisition. The optimal value of these parameters is sample dependant and different between MS^E and $HDMS^E$ modes. For *synapter* to function properly all acquisitions in the analysis have to be processed with the same thresholds, optimal for the mode identifications are transferred from (typically $HDMS^E$ mode). The user is expected to identify optimal parameters himself for every new sample type by repeatedly analysing a representative acquisition with different thresholds.

After the ions peaks have been determined and centroided, the ions representing charge states and isotopes of a peptide are collapsed into a single entity called EMRT (exact mass retention time pair). The EMRTs in low energy function represent unidentified peptides and are assigned peptides sequences

during database search. The total list of EMRTs can be found in the `pep3DAMRT.csv` file and it is one of the *synapter* input files for the runs used for quantitation (typically MS^E mode)

Prior to the database search, randomised entries are added to the database to allow PLGS to compute *protein false positive rate*. The randomised entries can either be added automatically or manually, using the *Randomise Databank* function in the *Databank admin tool*. To properly prepare the files for *synapter*, the user has to add randomised entries manually via *Databank admin tool*, since only then randomised entries identified in the database search will be displayed in the *csv* output. Figure **??** demonstrates how to create a randomised databank manually using one randomised entry per regular entry.

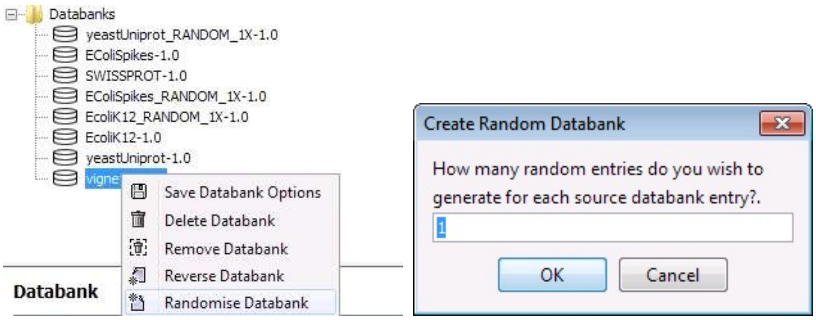
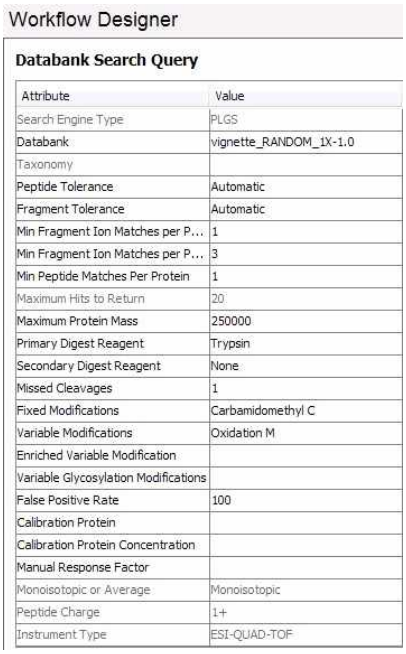


Figure 2: Databank creation in PLGS.



Attribute	Value
Search Engine Type	PLGS
Databank	vignette_RANDOM_1X-1.0
Taxonomy	
Peptide Tolerance	Automatic
Fragment Tolerance	Automatic
Min Fragment Ion Matches per P...	1
Min Fragment Ion Matches per P...	3
Min Peptide Matches Per Protein	1
Maximum Hits to Return	20
Maximum Protein Mass	250000
Primary Digest Reagent	Trypsin
Secondary Digest Reagent	None
Missed Cleavages	1
Fixed Modifications	Carbamidomethyl C
Variable Modifications	Oxidation M
Enriched Variable Modification	
Variable Glycosylation Modifications	
False Positive Rate	100
Calibration Protein	
Calibration Protein Concentration	
Manual Response Factor	
Monoisotopic or Average	Monoisotopic
Peptide Charge	1+
Instrument Type	ESI-QUAD-TOF

Figure 3: Databank search options.

The user is also expected to use a minimum of 1 fragment per peptide, 3 fragments per protein and 1 peptide per protein identification thresholds and 100% *False Positive Rate*[8] for protein identifica-

[8] This is erroneously termed false positive rate in the software and manuscript and should be considered a false discovery rate.

tion during database search for all of the acquisitions in the analysis as demonstrated in figure **??**. This allows to maximise the number of identified peptides from the randomised part of the database, needed to estimate peptide identifications statistics. The total list of identified peptides is given in `final_peptide.csv` files. A single `final_peptide.csv` file has to be supplied to *synapter* for every run in the analysis (for both identification and quantitation runs).

More details and screenshots are available in a separate document available at http://lgatto.github.com/synapter/.

2.2 HDMS^E/MS^E data analysis

The analysis of pairs of HDMS^E and MS^E data files is based on the following rationale – combine strengths or each approach by matching high quality HDMS^E identifications to quantified MS^E EMRTs applying the following algorithm:

1. Apply various peptide filters to HDMS^E and MS^E peptides to obtain two sets of reliably identified unique proteotypic peptides.
2. Use shared HDMS^E and MS^E peptides to model the deviations in retention time between the two mass spectrometer runs.
3. Optimise the parameters that will be used to optimally match all HDMS^E peptides and quantified MS^E EMRTs using a grid search.
4. Using the best parameters, match identified HDMS^E peptides to quantified MS^E EMRTs.

2.3 Different pipelines

Three different pipeline are available to the user:

2.3.1 Graphical user interface

The graphical interface is deprecated and will be removed in future versions. We encourage users to utilise the `synergise` instead.

A simple graphical interface (GUI) can be used to preform a complete data analysis. This pipeline is the most accessible for users that do not feel comfortable with command line interfaces (see below) and/or for a limited number of analysis to be run manually.

The GUI is a graphical layer between the user and the `synergise` function that will be described later. For more details on the underlying data processing and parameters that can be customised, read `?synergise`.

The graphical interface is shown on figure **??** and is started by calling the `synapterGUI()` function. The interface is composed of three tabs, that allow data input and analyses parameters customisation. The `synapterGUI` function itself takes one input parameter, `n`, that defines the number of identification/quantitation sets of files that the user wants to analyse. Each set is composed of one identification

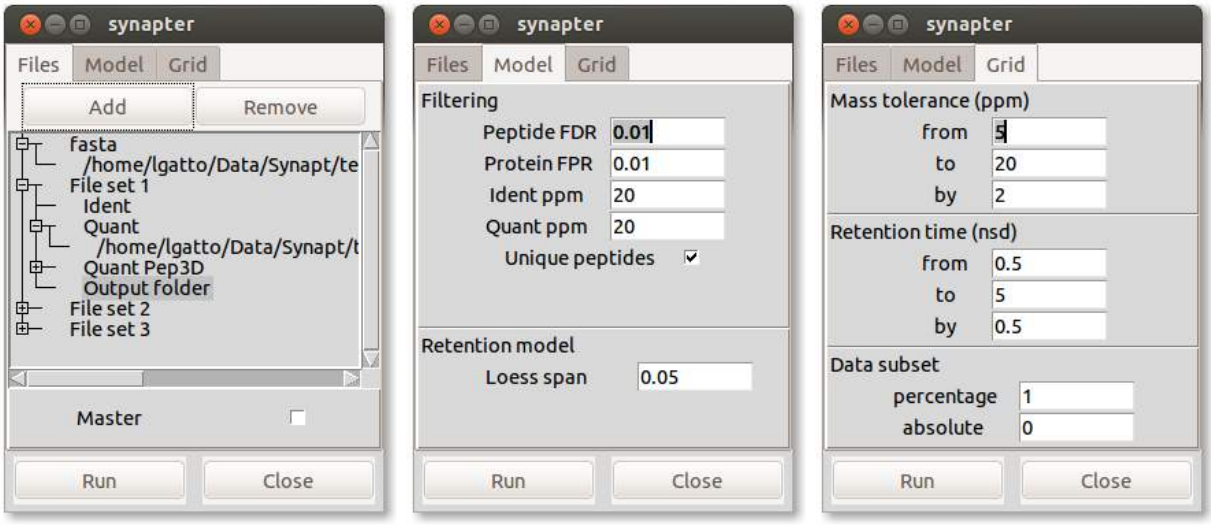


Figure 4: Screenshots of the 3 input tabs of the `synapterGUI` function. From left to right: (1) the data input tab, (2) the data filtering and retention time modelling tab and (3) the grid search tab.

final peptide file (typically HDMS^E), one quantitation final peptide file (typically MS^E), one quantitation Pep3D file (also MS^E) and one output directory. In addition, the user also needs to specify one single fasta file that will be used to filter proteotypic peptide. To perform 3 analysis, as illustrated on figure **??**, the function would be executed like `synapterGUI(n = 3)` or simply `synapterGUI(3)`.

Data input The first tab uses a tree structure to represent the input sets to be analysed. The unique fasta file is located at the very top of the hierarchy and each subsequent node (file set) can be opened and populated. Files are added by selecting the respective node, clicking the `Add` button and selecting the corresponding file using the file selection dialogue that opens. The file names then appear as new nodes (see for example the identification file in set 1) and can be removed with the `Remove` button.

The `Master` box needs to be checked if the identification inputs are *master* files (see section **??**).

Filtering and modelling The second tab allows to specify peptide filtering and retention time modelling parameters. Modelling accounts for systematic deviation in retention time for peptides between mass spectrometry runs (figure **??**) by fitting a curve through deviation of retention time vs retention time plot.

For a detailed description of the parameters and the processing pipeline, see the documentation of the `synergise` function.

Grid search Matching peptides to quantified EMRTs is done in the two dimensional retention time vs. precursor mass space. The optimal tolerances in both dimensions are estimated by a grid search that uses common identification/quantitation peptides. The size of the grid, i.e. the range of the retention time (`nsd`, number of standard deviations in the retention time model) and mass tolerance (`ppm`) to be searched can be defined here. In addition, it is possible to select a subset of the data to reduce search time. For a detailed description of the parameters, see the documentation of the `synergise` function.

Once all the input has been specified, pressing the `Run` button in the lower left corner of the GUI starts the *synapter* run: all `n` analyses are executed one after each other and a complete report in `html` as

```
> plotRt(ups25b, what = "model", nsd = 1)
```

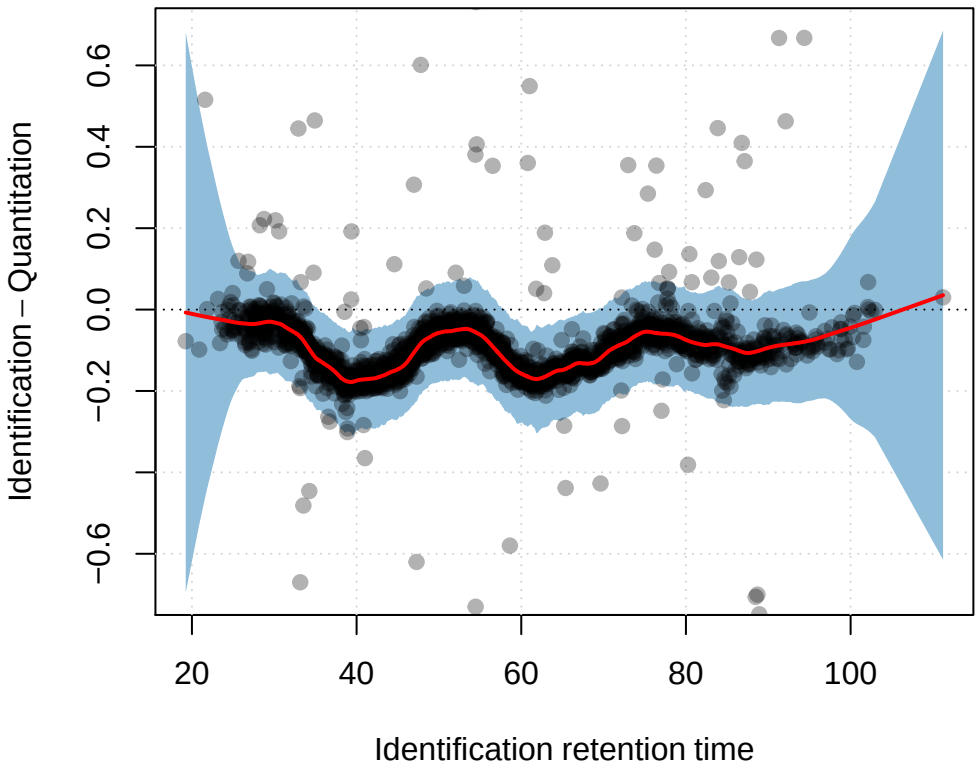


Figure 5: Figure illustrating retention time modelling between two runs in *synapter*, as generated by the `plotRt` function.

well as several result files are created in the respective output folders.

2.3.2 Wrapper function

The `synergise` function is a high level wrapper that implements a suggested analysis to combine two files (see next paragraph for details). A set of parameters can be passed, although sensible defaults are provided. While the analysis is executed, a html report is created, including all result files in text spreadsheet (`csv` format) and binary *R*output. This level allows easy scripting for automated batch analysis. Using data from the *synapterdata* package, the following code chunk illustrates the `synergise` usage. An example report can be found online at http://lgatto.github.com/synapter/.

```
> library(synapterdata)
> hdmsefile <- getHDMSeFinalPeptide()[2]
> basename(hdmsefile)
```

```
[1] "HDMSe_101111_25fmol_UPS1_in_Ecoli_04_IA_final_peptide.csv.gz"
```

```
> msefile <- getMSeFinalPeptide()[2]
> basename(msefile)
```

```
[1] "MSe_101111_25fmol_UPS1_in_Ecoli_03_IA_final_peptide.csv.gz"
```

```
> msepep3dfile <- getMSePep3D()[2]
> basename(msepep3dfile)
```

```
[1] "MSe_101111_25fmol_UPS1_in_Ecoli_03_Pep3DAMRT.csv.gz"
```

```
> fas <- getFasta()
> basename(fas)
```

```
[1] "EcoliK12_enolase_UPSsimga_NB.fasta"
```

```
> ## the synergise input is a (named) list of filenames
> input <- list(identpeptide = hdmsefile,
+               quantpeptide = msefile,
+               quantpep3d = msepep3dfile,
+               fasta = fas)
> ## a report and result files will be stored
> ## in the 'output' directory
> output <- tempdir()
> output
```

```
[1] "/tmp/RtmpB7ptd7"
```

```
> res <- synergise(filenames = input, outputdir = output)
```

```
> performance(res)
```

```
(S) Synapter: 4745 EMRTs uniquely matched.
(I) Ident: 5642 peptides.
(Q) Quant:   2685 peptides.
Enrichment (S/Q): 76.72%
Overlap:
   Q    S   QS
 240 2282 2445
```

See `?synergise` for details.

2.3.3 Detailed step-by-step analysis

The user can have detailed control on each step of the analysis by executing each low-level function manually. This pipeline, including generation of data containers (class instances) and all available operations are available in `?Synapter`. This strategy allows the maximum flexibility to develop new unexplored approaches.

2.4 Using master peptide files

While analysing one MS^E file against one single HDMS^E file increased the total number of reliably identified and quantified features compared to each single MS^E analysis, a better procedure can be applied when replicates are available. Consider the following design with two pairs of files: HDMS$^E_1$, MS$^E_1$, HDMS$^E_2$ and MS$^E_2$. The classical approach would lead to combining for example, HDMS$^E_1$ and MS$^E_1$ and HDMS$^E_2$ and MS$^E_2$. However, HDMS$^E_1$ – MS$^E_2$ and HDMS$^E_2$ – MS$^E_1$ would also be suitable, possibly leading to new identified and quantified features. Instead of repeating all possible combinations, which could hardly be applied for more replicates, we allow to merge HDMS$^E_1$ and HDMS$^E_2$ into a new *master* HDMS$^E_{12}$ and then using it to transfer identification to both MS^E runs. In addition to leading to a simpler set of analyses, this approach also allows to control the false positive rate during the HDMS^E merging (see section **??**). Such *master* HDMS^E files can be readily created with the `makeMaster` function, as described in section **??**.

We will use data from the *synapterdata* to illustrate how to create *master* files.

2.4.1 Choosing which HDMS^E files to combine

In a more complex design, a greater number of HDMS^E files might need to be combined. When combining files, one also accumulates false peptides assignments. The extent to which combining files increases new reliable identification at the cost of accumulating false assignments can be estimated with the `estimateMasterFdr` function.

To illustrate how FDR is estimated for *master* HDMS^E files, let's consider two extreme cases.

- In the first one, the two files (each with 1000 peptides filtered at an FDR of 0.01) to be combined are nearly identical, sharing 900 peptides. The combined data will have $900(shared) + 2 \times 100(unique)$ peptides and each file, taken separately is estimated to have $1000 \times 0.01 = 10$ false positive identifications. We thus estimate the upper FDR bound after merging the two files to be $\frac{20}{1100} = 0.0182$.
- In the second hypothetical case, the two files (again each with 1000 peptides filtered at a FDR of 0.01) to be combined are very different and share only 100 peptides. The combined data will have $100(shared) + 2 \times 900(unique)$ peptides and, as above, each file is estimated to have 10 false discoveries. In this case, we obtain an upper FDR bound of $\frac{20}{1900} = 0.0105$.

In general, the final false discovery for two files will be

$$FDR_{master} = \frac{nfd_1 + nfd_2}{union(peptides\ HDMS^E_1, peptides\ HDMS^E_2)}$$

where nfd_i is the number of false discoveries in HDMS^E file i. Note that we do not make any assumptions about repeated identification in multiple files here.

`estimateMasterFdr` generalised this for any number of HDMS^E files and indicates the best combination at a fixed user-specified `masterFdr` level. Mandatory input is a list of HDMS^E file names and a fasta database file name to filter non-unique proteotypic peptides.

The result of `estimateMasterFdr` stores the number of unique proteotypic peptides and FDR for all possible 57 combinations of 6 files. A summary can be printed on the console or plotted with `plot(cmb)` (see figure **??**).

```
> ## using the full set of 6 HDMSe files and a
> ## fasta database from the synapterdata package
> inputfiles <- getHDMSeFinalPeptide()
> fasta <- getFasta()
> cmb <- estimateMasterFdr(inputfiles, fasta, masterFdr = 0.02,
+                          verbose = FALSE)
> cmb

6 files - 57 combinations
Best combination: 4 5
 - 5730 proteotypic peptides
 - 6642 unique peptides
 - 0.017 FDR
```

The best combination can be extracted with the `bestComb` function.

```
> bestComb(cmb)

[1] 4 5
```

See `?estimateMasterFdr` and references therein for more details about the function and the returned object.

2.4.2 Generating a master file

Now that we have identified which files should be used to create the master file, we can directly pass the relevant identification files to the `makeMaster` function to generate the *master* file. The function has one mandatory input parameter, `pepfiles`, a list oh identification file names to be merged. The output is an object of class `MasterPeptides` that stores the relevant peptides from the original input files. The result can be saved to disk using `saveRDS` for further analysis, as described in section **??**.

```
> master <- makeMaster(inputfiles[bestComb(cmb)], verbose = FALSE)
> master

Object of class "MasterPeptides"
 1st Master [ 1 2 ] has 6699 peptides
 2nd Master [ 2 1 ] has 6709 peptides
 [1] HDMSe_111111_50fmol_UPS1_in_Ecoli_04_IA_final_peptide.csv.gz
 [2] HDMSe_111111_50fmol_UPS1_in_Ecoli_02_IA_final_peptide.csv.gz
```

More details can be found in the function documentation accessible with `?makeMaster`.

```
> plot(cmb)
```

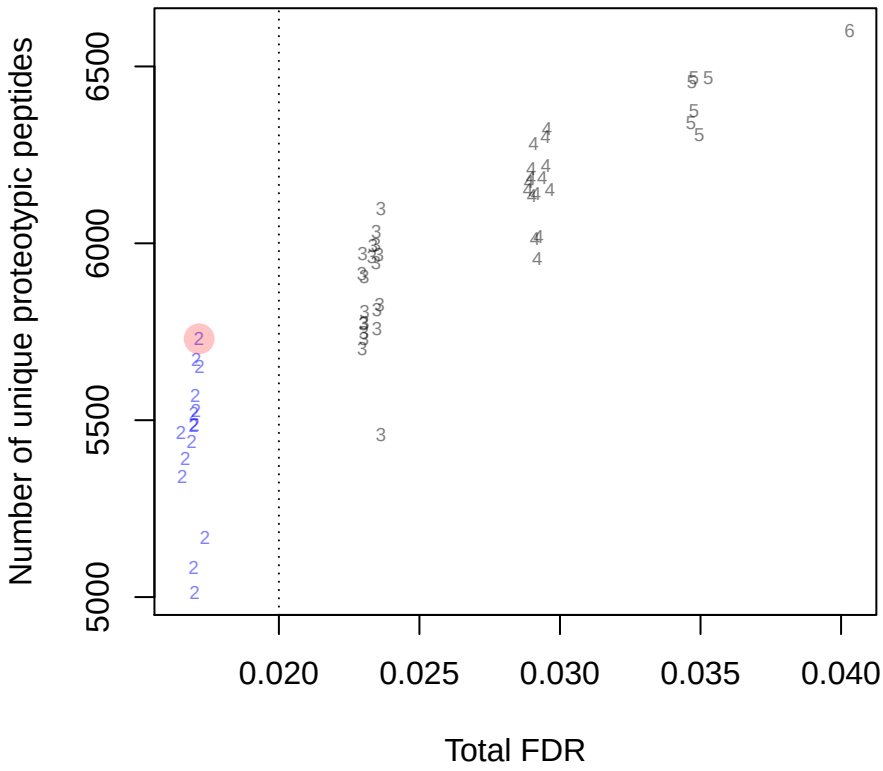


Figure 6: Figure illustrating the relation between the number of unique peptides in the combined HDMS^E file and the resulting false discovery rate. The symbols on the figure represent the number of files for that particular combination. The dotted line is the user defined threshold for the combined FDR (`masterFdr` parameter). The best combination, i.e the one that maximises the number of unique peptides while keeping the FDR below `masterFdr` is highlighted in red.

2.5 Summary

Two functions are needed to choose a set of IR files and create the *master* IR. One function enables to perform a complete identification transfer, eiter through a command line or graphical interface. Table **??** summarises all there is to know to utilise *synapter*'s functionality.

Function	Description
`synapterGUI()`	Opens the graphical user interface
`synergise`	Runs the complete identification transfer
`estimateMasterFdr`	Chooses which files to be used to create the *master* IR
`makeMaster`	Creates the *master* IR

Table 1: The *synapter* functions.

3 Analysing complete experiments

The functionality described in this section relies on the *MSnbase* package [**?**], which is installed by default with *synapter*. Please refer to the *MSnbase* Bioconductor web page[9], the associated vignettes and the respective manual pages for more details.

The *synapterdata* already provides preprocessed PLGS data. Six `Synapter` instances are available: 3 replicates (labelled `a`, `b` and `c`) of the Universal Proteomics Standard (UPS1) 48 protein mix at 25 fmol and 3 replicates at 50 fmol, in a constant *E. coli* background. The 6 files can be loaded in your working space with

```
> data(ups25a, ups25b, ups25c, ups50a, ups50b, ups50c)
```

3.1 Applying the Top 3 approach

We will start by describing the analysis of `ups25a` in details, and then show how to analyse all the runs using more compact code. The first step of our analysis is to convert the *synapter* object output (a `Synapter` instance), into a *MSnbase*-compatible object, called an `MSnSet`, that we will name `ms25a`. We can obtain a description of the `MSnSet` object by typing its name.

```
> ms25a <- as(ups25a, "MSnSet")
> class(ups25a)[1]

[1] "Synapter"

> class(ms25a)[1]

[1] "MSnSet"

> ms25a

MSnSet (storageMode: lockedEnvironment)
assayData: 5642 features, 1 samples
  element names: exprs
protocolData: none
phenoData: none
featureData
```

[9] http://bioconductor.org/packages/release/bioc/html/MSnbase.html

```
  featureNames: AALESTLAAITESLK IAAANVPAFVSGK ...
    NDSALGLFNGDIGIALDR (5642 total)
  fvarLabels: peptide.seq protein.Accession ... qval (20
    total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: No annotation
- - - Processing information - - -
 MSnbase version: 2.0.0
```

It contains quantitation information about 5642 peptides for 1 sample. In the code chunk below, we update the default sample name `Synapter1` with a more meaningful one.

```
> sampleNames(ms25a)

[1] "Synapter1"

> sampleNames(ms25a) <- "ups25a"
> sampleNames(ms25a)

[1] "ups25a"
```

Quantitative data and meta-data, which has been acquired by *synapter*, can be extracted with the `exprs` and `fData` methods.

```
> tail(exprs(ms25a))

                   ups25a
AFLNDK                 NA
IEAQLNDVIADLDAVR     1671
AVFNGLINVAQHAIK      1499
LEEVK                 664
VALQGNMDPSMLYAPPAR   2969
NDSALGLFNGDIGIALDR     NA

> tail(fData(ms25a)[, c(2,9)])

                   protein.Accession precursor.retT
AFLNDK                    MNME_ECODH       33.50725
IEAQLNDVIADLDAVR          MNME_ECODH       84.15609
AVFNGLINVAQHAIK         B1XFY9_ECODH       76.22252
LEEVK                   B1X7F0_ECODH       41.40945
VALQGNMDPSMLYAPPAR        DCUP_ECODH       67.60099
NDSALGLFNGDIGIALDR      B1XDM5_ECODH       97.65882

> ## all fetaure metadata
> fvarLabels(ms25a)

 [1] "peptide.seq"               "protein.Accession"
 [3] "protein.Description"       "protein.falsePositiveRate"
```

```
 [5] "peptide.matchType"         "peptide.mhp"
 [7] "peptide.score"             "precursor.mhp"
 [9] "precursor.retT"            "precursor.inten"
[11] "precursor.Mobility"        "spectrumID"
[13] "Intensity"                 "ion_ID"
[15] "ion_area"                  "ion_counts"
[17] "pval"                      "Bonferroni"
[19] "BH"                        "qval"
```

We will describe a how to process the data using a *Top 3* approach, where the 3 most intense peptides of each protein are used to compute the protein intensity, using the `topN` and `combineFeatures` methods. The former allows to extract the top most intense peptides (default `n` is 3) and remove all other peptides from the `MSnSet` object. The latter than aggregates the `n` most intense peptides per protein using a user-defined function (`sum`, below). Finally, we also scale protein intensity values depending on the actual number of peptides that have summed. This number of quantified peptides can be calculated (after `topN`, but before `combineFeatures`) with `nQuants`.

```
> ms25a <- topN(ms25a, groupBy = fData(ms25a)$protein.Accession, n = 3)
> nPeps <- nQuants(ms25a, groupBy = fData(ms25a)$protein.Accession)
> ms25a <- combineFeatures(ms25a, fData(ms25a)$protein.Accession,
+                          "sum", na.rm = TRUE, verbose = FALSE)
> head(exprs(ms25a))

           ups25a
6PGL_ECODH  71555
ABDH_ECODH  47542
ACCA_ECODH  38249
ACCD_ECODH  25615
ACP_ECODH   16388
APT_ECODH       0

> head(nPeps)

           ups25a
6PGL_ECODH      3
ABDH_ECODH      3
ACCA_ECODH      3
ACCD_ECODH      3
ACP_ECODH       1
APT_ECODH       0

> ## scale intensities
> exprs(ms25a) <- exprs(ms25a) * (3/nPeps)
> ## NaN result from the division by 0, when
> ## no peptide was found for that protein
> head(exprs(ms25a))
```

```
           ups25a
6PGL_ECODH  71555
ABDH_ECODH  47542
ACCA_ECODH  38249
ACCD_ECODH  25615
ACP_ECODH   49164
APT_ECODH     NaN
```

3.2 Batch processing

The code chunk below repeats the above processing for the other 5 UPS1/*E. coli* runs.

```
> nms <- c(paste0("ups", 25, c("b", "c")),
+          paste0("ups", 50, c("a", "b", "c")))
> tmp <- sapply(nms, function(.ups) {
+   cat("Processing", .ups, "... ")
+   ## get the object from workspace and convert to MSnSet
+   x <- get(.ups, envir = .GlobalEnv)
+   x <- as(x, "MSnSet")
+   sampleNames(x) <- .ups
+   ## extract top 3 peptides
+   x <- topN(x, groupBy = fData(x)$protein.Accession, n = 3)
+   ## calculate the number of peptides that are available
+   nPeps <- nQuants(x, fData(x)$protein.Accession)
+   ## sum top3 peptides into protein quantitation
+   x <- combineFeatures(x, fData(x)$protein.Accession,
+                        "sum", na.rm = TRUE, verbose = FALSE)
+   ## adjust protein intensity based on actual number of top peptides
+   exprs(x) <- exprs(x) * (3/nPeps)
+   ## adjust feature variable names for combine
+   x <- updateFvarLabels(x, .ups)
+   ## save the new MSnExp instance in the workspace
+   varnm <- sub("ups", "ms", .ups)
+   assign(varnm, x, envir = .GlobalEnv)
+   cat("done\n")
+ })
```

```
Processing ups25b ... done
Processing ups25c ... done
Processing ups50a ... done
Processing ups50b ... done
Processing ups50c ... done
```

We now have 6 `MSnSet` instances, containing protein quantitation for the 6 UPS/*E. coli* runs.

3.3 Combining data and filtering

We now want to filter data out based on missing quantitation data, retaining proteins that have been quantified in at a least two out of three replicates. Filtering based on missing data can be done using the `filterNA` method and a maximum missing data content as defined by `pNA`. Multiple `MSnSet` instances can be combined with the `combine` method, which is described in details in the `MSnbase-demo` vignette[10]. The 6 objects have appropriate distinct sample names and common feature (protein) names, which will be used to properly combine the quantitation data.

```
> ms25 <- combine(ms25a, ms25b)
> ms25 <- combine(ms25, ms25c)
> dim(ms25)

[1] 729   3

> ms25 <- filterNA(ms25, pNA = 1/3)
> dim(ms25)

[1] 709   3
```

Once combined and filtered, the 25 fmol group retains 709 entries with at least 2 out of 3 quantitation values, out of the 729 total number of proteins.

```
> ms50 <- combine(ms50a, ms50b)
> ms50 <- combine(ms50, ms50c)
> dim(ms50)

[1] 729   3

> ms50 <- filterNA(ms50, pNA = 1/3)
> dim(ms50)

[1] 709   3
```

Similarly, the 50 fmol group retains 709 entries with at least 2 out of 3 quantitation values, out of the 729 initial proteins.

We now combine the two subgroups into one `MSnSet` object that contains all 6 samples and filter for proteins that are observed in both groups, i.e retaining proteins with a maximum of 2/6 missing values. We also compute a summary table with the number of protein that have 4, 5, or 6 quantitation values across the 6 samples.

```
> msUps <- combine(ms25, ms50)
> msUps <- filterNA(msUps, pNA = 2/6)
> head(exprs(msUps))

           ups25a ups25b ups25c ups50a ups50b ups50c
6PGL_ECODH  71555  62114  60655  59920  56185  53874
ABDH_ECODH  47542  37805  36746  45570  43163  39506
```

[10] The vignette is accessible from within *R* with `vignette("MSnbase-demo", package = "MSnbase")`.

```
ACCA_ECODH  38249  31543  29570  30697  29656  27851
ACCD_ECODH  25615  22247  20295  22206  19698  19819
ACP_ECODH   49164 738365 706538 734425 712076 655842
AROB_ECODH   5442   4050   3684   4095   4500   3879
```

```
> table(apply(exprs(msUps), 1, function(.x) sum(!is.na(.x))))


  4   5   6
  6  25 674
```

We obtain a final data set containing 705 proteins. Finally, we normalise protein intensities in each sample to correct for experimental loading biases and pipetting errors. To do so, we compute 6 sample medians using all constant *E. coli* background proteins and divide each protein by its respective sample mean.

```
> ecoli <- -grep("ups$", featureNames(msUps))
> meds <- apply(exprs(msUps)[ecoli, ], 2, median, na.rm=TRUE)
> exprs(msUps) <- t(apply(exprs(msUps), 1, "/", meds))
```

This same procedure could be applied with a set of constant spikes to estimate absolute protein quantities.

3.4 Statistical analysis of differentially expressed proteins

The UPS1 spiked-in protein mix is composed of 48 proteins, 47 of which have been observed and quantified in our final data object. In this section, we will illustrate how to analyse the 705 proteins to extract those that show differences between the two groups and show that these candidates represent the UPS1 spikes.

The *R*environment and many of the available packages allow extremely powerful statistical analysis. In this document, we will apply a standard t-test on log_2 transformed data for convenience, to calculate p-value for individual proteins (pv variable below). For best performance with small number of samples and more complex designs, we recommend the Bioconductor *limma* package [**?**][11]. We then perform multiple comparison adjustment using the qvalue from the *qvalue* package, that implements the method from [**?**] (qv variable below). The *multtest* package provides several other p-value adjustment methods. We will also compute log_2 fold-changes and illustrate the results on a volcano plot (figure **??**). Figure **??** illustrates the UPS1 proteins and samples on a classical heatmap.

```
> ## use log2 data for t-test
> exprs(msUps) <- log2(exprs(msUps))
> ## apply a t-test and extract the p-value
> pv <- apply(exprs(msUps), 1 ,
+             function(x) t.test(x[1:3], x[4:6])$p.value)
> ## calculate q-values
```

[11] http://www.bioconductor.org/packages/release/bioc/html/limma.html

```
> library(qvalue)
> qv <- qvalue(pv)$qvalues
> ## calculate log2 fold-changes
> lfc <- apply(exprs(msUps), 1 ,
+              function(x) mean(x[1:3], na.rm=TRUE) - mean(x[4:6], na.rm=TRUE))
> ## create a summary table
> res <- data.frame(cbind(exprs(msUps), pv, qv, lfc))
> ## reorder based on q-values
> res <- res[order(res$qv), ]
> head(round(res, 3))
```

```
          ups25a ups25b ups25c ups50a ups50b ups50c pv    qv    lfc
P01112ups -0.053 -0.015 -0.001  1.072  1.118  1.080  0 0.000 -1.113
P00918ups -0.247 -0.201 -0.214  0.616  0.667  0.674  0 0.001 -0.873
P01008ups  0.112  0.106  0.178  1.075  1.132  1.090  0 0.001 -0.967
Q06830ups  0.174  0.118  0.156  1.073  1.095  1.100  0 0.002 -0.940
P10145ups -0.323 -0.332 -0.274  0.689  0.764  0.788  0 0.003 -1.057
P02788ups  0.564  0.647  0.601  1.494  1.532  1.532  0 0.003 -0.915
```

In the above example, quantitation values and statistics data are saved in a summary dataframe (`res`), that can be exported to a comma-separated spreadsheet with

```
> write.csv(res, file = "upsResults.csv")
```

A total 29 proteins show a statistically different pattern between the two groups, at a false discovery rate of 10%. Table **??** summarises the results for all UPS1 proteins.

```
null device
          1
```

	ups25a	ups25b	ups25c	ups50a	ups50b	ups50c	pv	qv	lfc
P01112ups	-0.05	-0.01	-0.00	1.07	1.12	1.08	0.00	0.00	-1.11
P00918ups	-0.25	-0.20	-0.21	0.62	0.67	0.67	0.00	0.00	-0.87
P01008ups	0.11	0.11	0.18	1.07	1.13	1.09	0.00	0.00	-0.97
Q06830ups	0.17	0.12	0.16	1.07	1.09	1.10	0.00	0.00	-0.94
P10145ups	-0.32	-0.33	-0.27	0.69	0.76	0.79	0.00	0.00	-1.06
P02788ups	0.56	0.65	0.60	1.49	1.53	1.53	0.00	0.00	-0.92
P02753ups	-1.90	-1.82	-1.91	-1.01	-0.92	-0.88	0.00	0.01	-0.94
P01375ups	0.81	0.93	0.96	1.82	1.76	1.69	0.00	0.01	-0.86
P69905ups	-1.44	-1.56	-1.51	-0.64	-0.58	-0.58	0.00	0.01	-0.91
P00167ups	0.87	0.89	0.97	1.92	1.96	1.94	0.00	0.01	-1.03
P12081ups	-0.09	0.10	-0.02	0.94	1.04	1.05	0.00	0.01	-1.01
P00709ups	-0.19	-0.31	-0.32	0.42	0.51	0.51	0.00	0.01	-0.75
O00762ups	0.43	0.27	0.26	1.09	1.21	1.21	0.00	0.01	-0.85
P05413ups	-0.17	-0.40	-0.28	0.58	0.68	0.69	0.00	0.02	-0.93
P00441ups	-0.23	-0.41	-0.39	0.29	0.41	0.41	0.00	0.03	-0.71
P04040ups	-0.07	0.14	0.21	1.09	1.25	1.24	0.00	0.03	-1.10
P02787ups	0.00	0.15	0.07	1.50	1.53	1.24	0.00	0.03	-1.35
P10636-8ups	0.73	0.53	0.56	1.49	1.50	1.58	0.00	0.03	-0.92
P06396ups	0.23	0.30	0.04	1.18	1.27	1.30	0.00	0.04	-1.06
P16083ups	-0.18	-0.41	-0.28	1.17	1.15	1.17	0.00	0.04	-1.45
P02768ups	0.55	0.34	0.40	1.35	1.43	1.41	0.00	0.04	-0.97
P01127ups	0.33	0.14	0.22	1.18	1.18	1.21	0.00	0.04	-0.96
P08758ups	0.27	0.09	0.08	1.14	1.18	1.16	0.00	0.05	-1.01
P00915ups	0.06	-0.18	0.06	1.10	1.14	1.16	0.00	0.06	-1.15
P15559ups	0.12	-0.09	-0.08	0.88	0.93	0.88	0.00	0.06	-0.91
P55957ups	-1.08	-1.46	-1.33	-0.35	-0.39	-0.18	0.00	0.06	-0.98
P62988ups	0.51	0.27	0.37	1.29	1.29	1.24	0.00	0.06	-0.89
P01031ups	-0.41	-0.65	-0.64	0.63	0.64	0.63	0.00	0.06	-1.20
P61626ups	-0.10	-0.36	-0.32	0.62	0.68	0.67	0.01	0.09	-0.92
P51965ups	-0.89	-1.18	-1.30	0.02	-0.04	-0.01	0.01	0.14	-1.11
P01344ups	-0.04	-0.40	-0.06	0.57	0.72	0.64	0.01	0.15	-0.81
P01579ups	-0.95	-0.72	-0.66	-0.26	-0.25	-0.17	0.02	0.20	-0.55
P41159ups	0.28	-0.21	-0.24	0.78	0.86	1.03	0.02	0.21	-0.94
P62937ups	-1.38	-0.69	-1.12	0.31	0.38	0.26	0.02	0.21	-1.38
P68871ups	-0.21	-0.44	-0.59	0.37	0.36	0.36	0.02	0.22	-0.78
P08263ups	-1.11	-0.64	-1.52	0.19	0.25	0.28	0.03	0.30	-1.33
P99999ups	-1.20	-1.99	-1.95	-0.90	-0.19	-0.84	0.04	0.32	-1.07
P10599ups	-0.88	-1.13	0.17	1.39	0.90	0.76	0.04	0.32	-1.63
P02144ups	-0.96	-0.12	-0.07	0.99	1.06	1.00	0.04	0.32	-1.40
P01133ups	-0.80	0.01	-0.38	0.53	0.56	0.50	0.06	0.43	-0.92
P02741ups	-0.27	-1.17	-1.09	0.31	0.32	0.30	0.06	0.43	-1.15
P61769ups	-0.66	-0.15	-0.16	0.72	0.38	-0.03	0.07	0.47	-0.68
P63165ups	-0.98	0.08	0.22	1.07	1.12	1.11	0.07	0.47	-1.33
O76070ups	-0.46	0.25	0.33	0.88	0.80	0.94	0.08	0.48	-0.83
P06732ups	1.50	0.54	0.56	1.32	1.36	1.38	0.27	0.50	-0.48
P63279ups	-1.86	-3.08	-1.58	-0.64	-0.68	-0.69	0.08	0.50	-1.50
P09211ups	1.73	-1.86	-1.84	-0.76	-0.33	-0.65	0.95	0.59	-0.08

Table 2: UPS1 proteins.

```
> plot(res$lfc, -log10(res$qv),
+      col = ifelse(grepl("ups$", rownames(res)),
+        "#4582B3AA",
+        "#A1A1A180"),
+      pch = 19,
+      xlab = expression(log[2]~fold-change),
+      ylab = expression(-log[10]~q-value))
> grid()
> abline(v = -1, lty = "dotted")
> abline(h = -log10(0.1), lty = "dotted")
> legend("topright", c("UPS", "E. coli"),
+        col = c("#4582B3AA", "#A1A1A1AA"),
+        pch = 19, bty = "n")
```

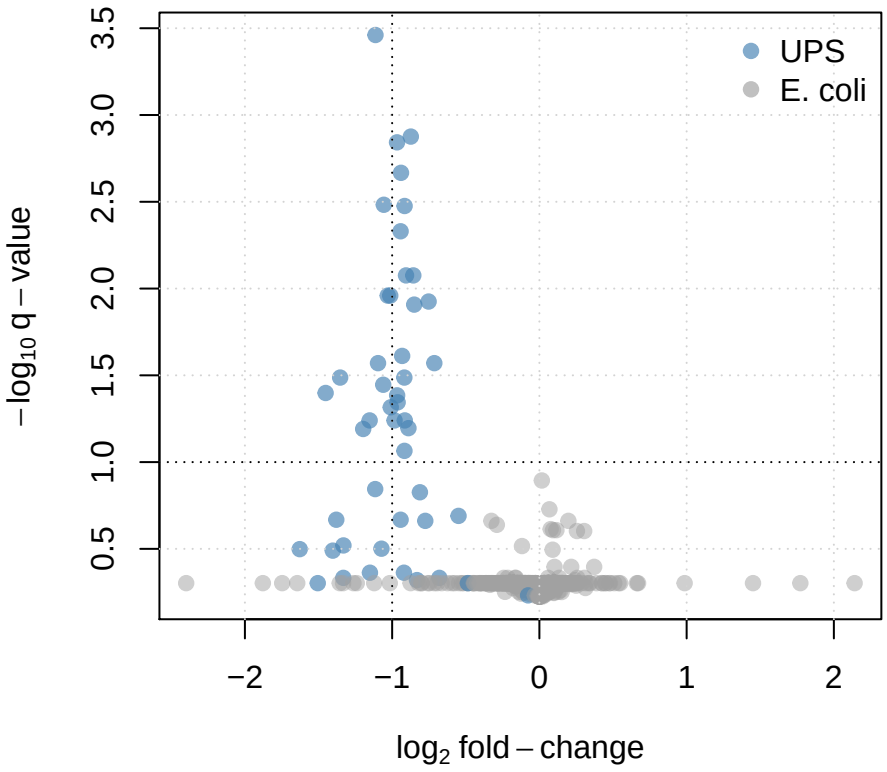


Figure 7: On the volcano plot, each protein is represented by a dot and positioned according to its log_2 fold-change along the x axis and $-log_{10}$ of its q-value along the y axis. Proteins with large fold-changes are positioned on the sides of the plot, while proteins with low q-values are at the top of the figure. The most promising candidates are this located on the top corners. In our case, the UPS proteins (in blue) have $log2$ fold-changes around -1 (vertical dotted line), as expected. The horizontal dotted line represents a q-value threshold of 0.10.

```
> heatmap(exprs(msUps)[grep("ups",featureNames(msUps)), ])
```

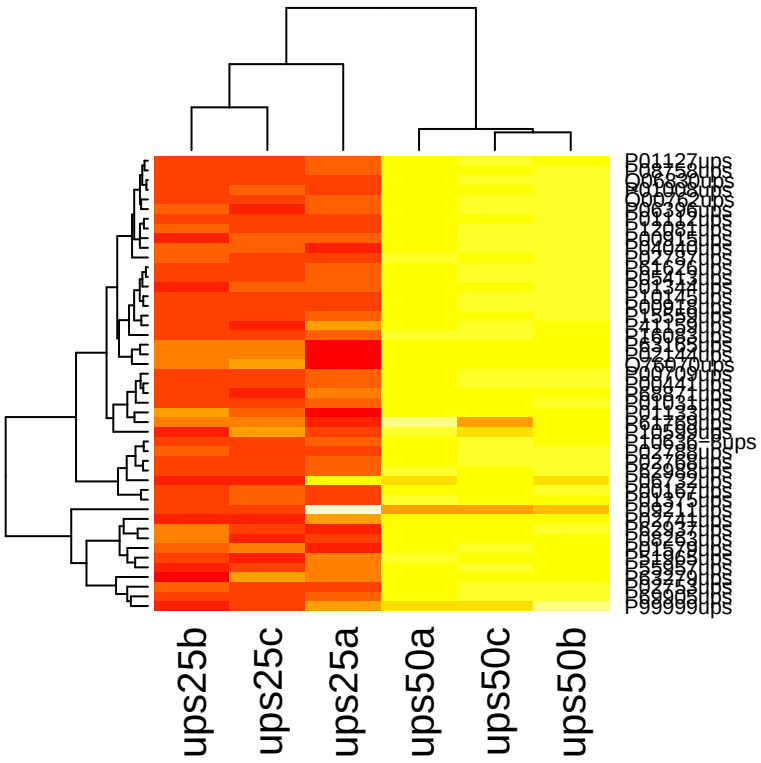


Figure 8: A heatmap of all UPS1 proteins present in the final data set.

4 Session information

All software and respective versions used to produce this document are listed below.

- R version 3.3.1 (2016-06-21), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, BiocParallel 1.8.0, MSnbase 2.0.0, ProtGenerics 1.6.0, Rcpp 0.12.7, knitr 1.14, mzR 2.8.0, qvalue 2.6.0, synapter 1.16.0, synapterdata 1.11.0, xtable 1.8-2
- Loaded via a namespace (and not attached): BiocInstaller 1.24.0, BiocStyle 2.2.0, Biostrings 2.42.0, IRanges 2.8.0, MALDIquant 1.15, MASS 7.3-45, Matrix 1.2-7.1, RColorBrewer 1.1-2, S4Vectors 0.12.0, XML 3.98-1.4, XVector 0.14.0, affy 1.52.0, affyio 1.44.0, cleaver 1.12.0, codetools 0.2-15, colorspace 1.2-7, digest 0.6.10, doParallel 1.0.10, evaluate 0.10, foreach 1.4.3, formatR 1.4, ggplot2 2.1.0, grid 3.3.1, gtable 0.2.0, highr 0.6, hwriter 1.3.2, impute 1.48.0, iterators 1.0.8, lattice 0.20-34, limma 3.30.0, magrittr 1.5, multtest 2.30.0, munsell 0.4.3, mzID 1.12.0, pcaMethods 1.66.0, plyr 1.8.4, preprocessCore 1.36.0, reshape2 1.4.1, scales 0.4.0, splines 3.3.1, stats4 3.3.1, stringi 1.1.2, stringr 1.1.0, survival 2.39-5, tools 3.3.1, vsn 3.42.0, zlibbioc 1.20.0