Tutorial 6: Working with spectral-level data

DFM & PL

2022-03-12

Contents

Installing the protlocassign package

This package contains functions to identify outlier spectra and peptides and then to compute weighted averages of the spectral profiles to produce a single profile for each protein. Before we describe how to use these functions, we must cover some preliminary steps needed to use them.

Start by installing the devtools package from CRAN, by typing:

```
install.packages("devtools")
```

If not yet installed, install the `protlocassign` package from the github repository by typing

```
BiocManager::install("protlocassign")
```

This will make the programs and data sets available, as well as installing needed libraries

The package requires that you attach these libraries:

```
library(outliers)
library(lme4)
```

These libraries include the `lme4` package, which implements a set of routines for working with nested data. In this case, this specifically applies to the nesting of spectra within peptides and peptides within proteins. The `outliers` package implements a score-based outlier detection routine, which is one option we provide.

The package `protlocassign` is required:

```
library(protlocassign)
```

Reading in spectral-level data from an external file

In this section we describe how to access the spectral-level data from the AT5-TMT2 experiment described in Tannous et al. 2020. The instructions here also apply to reading in another data set of interest.

The external data file must have the following structure. For each spectrum, the first and second columns must be a protein and peptide identifier. The next columns can contain ancillary information (e.g., a unique spectral identifier, the position of a peptide within a protein, etc) followed by NSA profiles. For example, suppose your data is in a comma-delimited file "myProteinData.csv" in the subdirectory "C:\temp\myProteinInput". You may read it into R as follows; the `str` command provides a look at the structure of the data that you have read in.

```
mySpectraData <- read.csv("C:\\temp\\mySpectraInput\\mySpectraData.txt")
str(mySpectraData, strict.width="cut", width=65)
```

If the full Tannous AT5 TMT-MS2 data are desired, they may be downloaded from an external file stored on the MassIVE Center for Quantatiative Mass Spectrometry housed at the University of California at San Diego. The required code is as follows:

```
urlQuantPSM <- "https://massive.ucsd.edu/ProteoSAFe/DownloadResultFile?file=f.MSV000083848/updates/2022
QuantPSM <- read.csv(urlQuantPSM)
str(QuantPSM, strict.width="cut", width=65)
```

If the protein names (which are in column 1) are prepended by a single quote (this prevents Excel from interpreting some protein names as dates), it may be removed by using the `sub` function to substitute the initial quote (denoted by "^"') with an empty character (" ") as shown here:

```
QuantPSM[,1] <- sub("^'", "", QuantPSM[,1])
```

The function `proteinDataPrep` is available to aid in formatting a file in the required form. This takes input data, sorts it by the first and second columns (which are renamed `prot` and `peptide`, the latter being a concatenation of the protein name and peptide sequence), appends all remaining columns (retaining their names), and generates a unique numerical ID for prot and peptide. We specify `numRefCols=5` because that is the number of columns immediately preceding the profile data, and `numDataCols=9` because that is the number of fractions in each profile.

```
spectraNSA_AT5tmtMS2 <- proteinDataPrep(QuantPSM, numRefCols=5,
                                        numDataCols=9)
str(spectraNSA_AT5tmtMS2, strict.width="cut", width=65)
```

The data frame `spectraNSA_AT5tmtMS2` contains the full data set. For the remainder of this tutorial, we may simplify the name as follows:

```
spectraNSA <- spectraNSA_AT5tmtMS2
```

For the purposes of this tutorial, we have provided a test data set `spectraNSA_test` consisting of the spectra from genes TLN1, TLN2, and all of the Jadot marker proteins (listed in the included data frame `markerListJadot`). We use this data as follows:

```
data(spectraNSA_test)
spectraNSA <- spectraNSA_test
```

Note that in some cases, one may want to change the peptide identifier to include additional information other than the actual sequence such as presence of post-translational modifications. In this case, one would create a new variable `PepMod` prior to running `proteinDataPrep`, and then insert that new column after the first column, which contains the protein name.

Note that in `spectraNSA_AT5tmtMS2`, the first five columns in the data frame contain information associated with the spectra assignment (protein identifier, peptide identifier, unique spectral identifier, location of peptide in protein, and post-translational/chemical modifications), while the next nine columns contain normalized specific amount data. The final two columns contain numbered lists of distinct proteins and peptides; these two columns are needed by the outlier detection functions described in the next section. The last section of this tutorial explains how to read in data from an external text file and describes a function `proteinDataPrep` that aids in putting the data into the above-described format.

Removal of outliers

The first step is an outlier screen, which is performed at the spectrum level. For each spectrum, the normalized specific amount for each of the nine fractions is first log2 transformed via $y = log2(c + \epsilon)$, where ϵ is an estimate of the background intensity in the abundance estimate c. We estimated ϵ as the median of the normalized specific amount of an isobaric labeling channel that lacks the bacterial standard DrR57, which is absent in mammalian cells, and which was added to all other channels. For the TMT-MS2 experiment we found $\epsilon = 0.029885$. The package provides a choice of two different outlier rejection methods that can be invoked to process the data. The "boxplot" method rejects any value that exceeds a specified multiple of the interquartile range below the first or above the third quartile. This method was used in Tannous et al. (2020) using three times the interquartile range, which we invoke here by specifying `outlierMeth="boxplot"` and `range=3`. The "scores" method used in Jadot et al (2017) calculates normal scores (differences between each value and the mean of those values, divided by their standard deviation) and rejects values in the tails of a standard normal distribution for a specified percentile, such as 0.99. It may be invoked by specifying `outlierMeth="scores"` and `proba=0.99`. Note that the outlier rejection procedure is conducted on each channel, and the number of outliers for each spectrum are reported in the variable `outlier.num.spectra`, which can range from zero (no outliers) to the number of fractions (in this case 9). This process is repeated for every peptide from every protein. We reject any spectrum where `outlier.num.spectra > 0`.

We first process the data to identify outliers when grouping spectra to peptides using the function `outlierFind` and specifying that the outlier level is "peptide", meaning that profiles with outlier spectra within peptides (accepting all peptides unless they are composed entirely of spectra with outliers) will be flagged. The arguments `numRefCols = 5` and `numDataCols = 9` are used to identify position of the nine-element profile in the data frame. The argument `randomError=TRUE` (which is the default) specifies that a small random number (uniform distribution ranging from zero to `eps`) will be added to profile elements that take the value zero. This addresses a problem that arises when there are a large number of zero-valued elements, i.e., both the first and third quartile, in which case the interquartile range will be zero. Without adding these random small amounts, too many "outliers" would be detected. We specify a random number seed `setSeed` so that the results are exactly reproducible. (The `setSeed` parameter is ignored if cpus=1; in that case, use the `set.seed` function if reproducibility is required.) Note that routines to identify outliers with our example data set will take several minutes to complete. When the option `cpus` is set to 2 or greater the function uses the `BiocParallel` library from Bioconductor to carry out parallel processing. If `cpus=1` then no parallel processing is done. The following code will install the package if it is not already present:

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
```

```
if (!require("BiocParallel", quietly = TRUE))
    BiocManager::install("BiocParallel")
```

```
eps <- 0.029885209
library(BiocParallel)
flagSpectraBox <- outlierFind(protClass=spectraNSA,
              outlierLevel="peptide", numRefCols=5, numDataCols=9,
              outlierMeth="boxplot", range=3, eps=eps,
              randomError=TRUE, setSeed=17356, cpus=4)
```

Alternatively, we can specify using the "scores" method as follows. Note that the argument "range" is only used when `outlierMeth="boxplot"`, and ignored otherwise. Similarly, the "proba" argument is only used when `outlierMeth="scores"`

```
flagSpectraScore <- outlierFind(protClass=spectraNSA,
                outlierLevel="peptide", numRefCols=5, numDataCols=9,
                outlierMeth="scores", proba=0.99,
                eps=eps, randomError=TRUE, setSeed=27681, cpus=4)
```

We then examine the output files from the boxplot and scores methods:

```
str(flagSpectraBox, strict.width="cut", width=65)
```

```
#> 'data.frame':    6696 obs. of  17 variables:
#>  $ prot                           : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                        : chr  "ACP2::CPLQDFLR" "ACP2"..
#>  $ SpectrumID                     : chr  "TGR_03069MS2_[Node_02"..
#>  $ PeptidesStartPositionInProtein: int  349 349 349 54 54 54 54..
#>  $ modifications                  : chr  "57.02147@C1,229.16293"..
#>  $ outlier.num.spectra            : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ N                              : num  0.0273 0.0582 0.038 0.0..
#>  $ M                              : num  0.0311 0.1115 0.0339 0...
#>  $ L1                             : num  0.1291 0.1283 0.1078 0...
#>  $ L2                             : num  0.0759 0.0926 0.0937 0...
#>  $ P                              : num  0.0157 0.0273 0.0519 0...
#>  $ S                              : num  0.0138 0.0186 0.0332 0...
#>  $ Nyc1                           : num  0.1307 0.1126 0.128 0.1..
#>  $ Nyc2                           : num  0.515 0.38 0.447 0.476 ..
#>  $ Nyc3                           : num  0.0616 0.0707 0.0665 0...
#>  $ protId                         : int  218 218 218 218 218 218..
#>  $ pepId                          : int  1700 1700 1700 1701 170..
```

```
str(flagSpectraScore, strict.width="cut", width=65)
```

```
#> 'data.frame':    6696 obs. of  17 variables:
#>  $ prot                           : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                        : chr  "ACP2::CPLQDFLR" "ACP2"..
#>  $ SpectrumID                     : chr  "TGR_03069MS2_[Node_02"..
#>  $ PeptidesStartPositionInProtein: int  349 349 349 54 54 54 54..
#>  $ modifications                  : chr  "57.02147@C1,229.16293"..
#>  $ outlier.num.spectra            : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
#>  $ N                             : num  0.0273 0.0582 0.038 0.0..
#>  $ M                             : num  0.0311 0.1115 0.0339 0...
#>  $ L1                            : num  0.1291 0.1283 0.1078 0...
#>  $ L2                            : num  0.0759 0.0926 0.0937 0...
#>  $ P                             : num  0.0157 0.0273 0.0519 0...
#>  $ S                             : num  0.0138 0.0186 0.0332 0...
#>  $ Nyc1                          : num  0.1307 0.1126 0.128 0.1..
#>  $ Nyc2                          : num  0.515 0.38 0.447 0.476 ..
#>  $ Nyc3                          : num  0.0616 0.0707 0.0665 0...
#>  $ protId                        : int  218 218 218 218 218 218..
#>  $ pepId                         : int  1700 1700 1700 1701 170..
```

We can determine the total number of spectra as well as the number of acceptable (e.g. non-outlier) and outlier spectra for both methods using the following statements:

```
length(flagSpectraBox$outlier.num.spectra)    # total number of spectra
```

```
#> [1] 6696
```

```
sum(flagSpectraBox$outlier.num.spectra == 0)  # number of acceptable spectra
```

```
#> [1] 6253
```

```
sum(flagSpectraBox$outlier.num.spectra != 0)  # number of outlier spectra
```

```
#> [1] 443
```

Similarly, we can determine the total number of spectra and the number of acceptable spectra for the scores method using the following statements:

```
length(flagSpectraScore$outlier.num.spectra)    # total number of spectra
```

```
#> [1] 6696
```

```
sum(flagSpectraScore$outlier.num.spectra == 0)  # number of acceptable spectra
```

```
#> [1] 6109
```

```
sum(flagSpectraScore$outlier.num.spectra != 0)  # number of outlier spectra
```

```
#> [1] 587
```

Note that in both examples shown above, the `randomError` argument was used (`randomError=TRUE`). The reason for invoking this argument is illustrated in the two examples below where we set the argument to `False` or `True` when we use the boxplot method with a very large range (`range=110`) where we intuitively would expect that there would be very few outliers. However, if we use the `randomError=FALSE` argument we find that there are 56 outliers:

```
flagSpectraBoxRange110ranErrF <- outlierFind(protClass=spectraNSA,
              outlierLevel="peptide", numRefCols=5, numDataCols=9,
              outlierMeth="boxplot", range=110, eps=eps,
              randomError=FALSE, setSeed=NULL, cpus=4)
sum(flagSpectraBoxRange110ranErrF$outlier.num.spectra != 0)
```

```
#> [1] 2
```

We have identified 2 outliers because, as discussed above, there are some peptides with enough 0 values that the interquartile range is 0. We can correct this by specifying `randomError=TRUE`, which reduces the number of outliers. Note in the example below there are no outliers. With repeated runs of `outlierFind`, unless the same random number seed is set, the random error will change, which will result in slightly varying outcomes.

```
flagSpectraBoxRange110ranErrT <- outlierFind(protClass=spectraNSA,
              outlierLevel="peptide", numRefCols=5, numDataCols=9,
              outlierMeth="boxplot", range=110, eps=eps,
              randomError=TRUE, setSeed=652908, cpus=4)
sum(flagSpectraBoxRange110ranErrT$outlier.num.spectra != 0)
```

```
#> [1] 0
```

The procedure described is useful in eliminating outlying spectra for estimates of peptide profiles. The next step is to create mean profiles for each peptide, and also to identify outlying peptide profiles.

Compute means for each protein-peptide combination

The function `profileSummarize` with option `GroupBy="peptideId"` computes the mean profile of each peptide, eliminating spectra that have been identified as outliers. The input data, `flagSpectraBox` contains six reference columns (see above), specified by `numRefCols=6`. A portion of these can be kept and carried over to the output file, but note that since the function here is producing a profile for each peptide from its relevant component spectra, some of the reference columns from the input data may not be appropriate. Here, the argument `refColsKeep=c(1,2,4)` causes the function to retain reference columns 1, 2, and 4, which correspond to "prot", "peptide", and "PeptidesStartPositionInProtein". If a field is kept which has multiple values across all spectra that contribute to a given peptide, only the single value from the first of these spectra is used. The argument `refColsKeep` is only used in the case where `GroupBy` is set to `peptideId`, and if not specified, the default is to keep only the first two columns. We use the argument `outlierExclude="spectra"` to indicate that, when computing mean profiles, we exclude spectra that are identified as outliers within their respective peptides. Note that in the output, `Nspectra` reports the number of spectra (in this case, all non-outlier spectra) used to calculate the mean peptide profile.

```
pepProfiles <- profileSummarize(protsCombineCnew=flagSpectraBox,
               numRefCols=6, numDataCols=9, refColsKeep=c(1,2,4),eps=eps,
               GroupBy="peptideId", outlierExclude="spectra", cpus=4)
str(pepProfiles, strict.width="cut", width=65)
```

```
#> 'data.frame':    944 obs. of  16 variables:
#>  $ prot                          : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                       : chr  "ACP2::CPLQDFLR" "ACP2"..
#>  $ PeptidesStartPositionInProtein: int  349 54 71 357 406 267 9..
```

```
#>  $ N                              : num  0.0407 0.0439 0.0642 0...
#>  $ M                              : num  0.0528 0.0413 0.0659 0...
#>  $ L1                             : num  0.1232 0.1074 0.0964 0...
#>  $ L2                             : num  0.0884 0.0784 0.1029 0...
#>  $ P                              : num  0.0303 0.0371 0.0677 0...
#>  $ S                              : num  0.0216 0.0559 0.0724 0...
#>  $ Nyc1                           : num  0.125 0.119 0.138 0.13 ..
#>  $ Nyc2                           : num  0.451 0.458 0.325 0.45 ..
#>  $ Nyc3                           : num  0.0671 0.0589 0.0677 0...
#>  $ Nspectra                       : num  3 4 14 4 6 1 7 2 2 1 ...
#>  $ Npep                           : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ protId                         : num  218 218 218 218 218 218..
#>  $ pepId                          : num  1700 1701 1702 1703 170..
```

Next, using this output, we use the function `outlierFind` to identify peptide outliers within proteins. We do this by specifying `outlierLevel` to be "protein" and do not need to specify that we use the default boxplot method with `range=3` and the `randomError=TRUE`. Note that in this procedure, all peptides in a given protein are weighted equally, regardless of how many non-outlier spectra are used to determine the average peptide profile.

```
flagPepsBox <- outlierFind(protClass=pepProfiles,
                    outlierLevel="protein",
                    numRefCols=3, numDataCols=9, eps=eps,
                    setSeed=823537, cpus=4)
str(flagPepsBox, strict.width="cut", width=65)
```

```
#> 'data.frame':    944 obs. of  17 variables:
#>  $ prot                           : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                        : chr  "ACP2::CPLQDFLR" "ACP2"..
#>  $ PeptidesStartPositionInProtein: int  349 54 71 357 406 267 9..
#>  $ outlier.num.peptides           : num  0 0 0 0 0 0 0 0 1 1 ...
#>  $ N                              : num  0.0407 0.0439 0.0642 0...
#>  $ M                              : num  0.0528 0.0413 0.0659 0...
#>  $ L1                             : num  0.1232 0.1074 0.0964 0...
#>  $ L2                             : num  0.0884 0.0784 0.1029 0...
#>  $ P                              : num  0.0303 0.0371 0.0677 0...
#>  $ S                              : num  0.0216 0.0559 0.0724 0...
#>  $ Nyc1                           : num  0.125 0.119 0.138 0.13 ..
#>  $ Nyc2                           : num  0.451 0.458 0.325 0.45 ..
#>  $ Nyc3                           : num  0.0671 0.0589 0.0677 0...
#>  $ Nspectra                       : num  3 4 14 4 6 1 7 2 2 1 ...
#>  $ Npep                           : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ protId                         : num  218 218 218 218 218 218..
#>  $ pepId                          : num  1700 1701 1702 1703 170..
```

We determine the total number of peptides and the number of acceptable and outlier peptides using the following statements:

```
length(flagPepsBox$outlier.num.peptides)
```

```
#> [1] 944
```

```
sum(flagPepsBox$outlier.num.peptides == 0)
```

```
#> [1] 857
```

```
sum(flagPepsBox$outlier.num.peptides != 0)
```

```
#> [1] 87
```

We can now merge the indicator of peptide outlier status into the full spectral level file. Here, we create a new data frame, `flagSpectraPeps`, by adding `outlier.num.peptides` (column 4 from `flagPepsBox`) to `flagSpectraBox`, merging on pepId (column 17 from `flagPepsBox`).

```
data.frame(names(flagPepsBox))
```

```
#>                names.flagPepsBox.
#> 1                            prot
#> 2                         peptide
#> 3  PeptidesStartPositionInProtein
#> 4            outlier.num.peptides
#> 5                               N
#> 6                               M
#> 7                              L1
#> 8                              L2
#> 9                               P
#> 10                              S
#> 11                           Nyc1
#> 12                           Nyc2
#> 13                           Nyc3
#> 14                       Nspectra
#> 15                           Npep
#> 16                         protId
#> 17                          pepId
```

```
flagSpectraPeps <- merge(x=flagSpectraBox,
                         y=flagPepsBox[,c(4,17)], by="pepId")
str(flagSpectraPeps,  strict.width="cut", width=65)
```

```
#> 'data.frame':    6696 obs. of  18 variables:
#>  $ pepId                          : int  1700 1700 1700 1701 170..
#>  $ prot                           : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                        : chr  "ACP2::CPLQDFLR" "ACP2"..
#>  $ SpectrumID                     : chr  "TGR_03069MS2_[Node_02"..
#>  $ PeptidesStartPositionInProtein: int  349 349 349 54 54 54 54..
#>  $ modifications                  : chr  "57.02147@C1,229.16293"..
#>  $ outlier.num.spectra            : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ N                              : num  0.0273 0.0582 0.038 0.0..
#>  $ M                              : num  0.0311 0.1115 0.0339 0...
#>  $ L1                             : num  0.1291 0.1283 0.1078 0...
#>  $ L2                             : num  0.0759 0.0926 0.0937 0...
#>  $ P                              : num  0.0157 0.0273 0.0519 0...
```

```
#>  $ S                              : num  0.0138 0.0186 0.0332 0...
#>  $ Nyc1                           : num  0.1307 0.1126 0.128 0.1..
#>  $ Nyc2                           : num  0.515 0.38 0.447 0.476 ..
#>  $ Nyc3                           : num  0.0616 0.0707 0.0665 0...
#>  $ protId                         : int  218 218 218 218 218 218..
#>  $ outlier.num.peptides           : num  0 0 0 0 0 0 0 0 0 0 ...
```

We determine the number of spectra in the full data set and the number of acceptable spectra from acceptable peptides using the following statements:

```
length(flagSpectraPeps$outlier.num.peptides)
```

```
#> [1] 6696
```

```
sum({flagSpectraPeps$outlier.num.spectra == 0} &
      {flagSpectraPeps$outlier.num.peptides == 0})
```

```
#> [1] 6033
```

We now calculate the mean profile for each protein, using a random effects linear model where spectra are nested within peptides. In the three examples listed below, we examine cases where we: (1) reject the contribution of any outlying spectra and peptides; (2) reject the contribution of just outlier spectra within peptides; and (3) we do not reject any peptides or spectra. Here, the `outlierExclude` argument overrides the outlier specifications in the `flagSpectraPeps` file. The resulting file contains the protein profile, indicating the number of peptides and spectra used to determine each protein profile using the rules above. The root portion of the name `protNSA` denotes that these values are normalized specific amounts (NSAs) for protein profiles.

The `lmer` random effects function within `profileSummarize` will generate error messages for some channels within some proteins due to numerical singularity issues. To minimize these errors, the function requires two conditions to be met before calling the `lmer` function: There must be at least eight spectra and four peptides, and the ratio of numbers of spectra to peptides must exceed 4. If these conditions are not met the function calculates a standard (non-nested) average of all spectra to obtain a mean profile. Otherwise, the `lmer` random effects model is fitted. In some cases, for a given protein and channel, the `lmer` routine fails and reports a singularity in the model fit. In these cases, `profileSummarize` automatically re-calculates the profile average using standard (non-nested) averaging. If the option `singularList=TRUE` is specified, the function will list the specific protein and channel number (which here range from 1 to 9) where the `lmer` routine encountered the singularity. For example, "ACP2" (a lysosomal protein) has 41 spectra and 8 peptides. The `lmer` procedure converged for the first eight fractions but failed for the ninth one (Nyc3). The default is `singularList=FALSE`.

```
protNSA_1 <-
          profileSummarize(protsCombineCnew=flagSpectraPeps,
          numRefCols=7, numDataCols=9, eps=eps, GroupBy="protId",
          outlierExclude="spectraAndPeptide", cpus=4, singularList=TRUE)
```

```
#> boundary (singular) fit: see help('isSingular')
#> protein MDH2  channel 5
#> boundary (singular) fit: see help('isSingular')
#> protein MDH2  channel 9
```

```
#> boundary (singular) fit: see help('isSingular')
#> protein GLB1  channel 1
#> boundary (singular) fit: see help('isSingular')
#> protein GLB1  channel 2
#> boundary (singular) fit: see help('isSingular')
#> protein GLB1  channel 4
#> boundary (singular) fit: see help('isSingular')
#> protein HEXA  channel 2

#> boundary (singular) fit: see help('isSingular')
#> protein RPN2  channel 2

#> boundary (singular) fit: see help('isSingular')
#> protein ACP2  channel 9
#> boundary (singular) fit: see help('isSingular')
#> protein CTSD  channel 1
```

```
str(protNSA_1,  strict.width="cut", width=65)
```

```
#> 'data.frame':    40 obs. of  12 variables:
#>  $ prot    : chr  "ACP2" "AIF1" "ATP1A1" "B4GALT1" ...
#>  $ N       : num  0.0483 NA 0.1169 0.0542 0.1026 ...
#>  $ M       : num  0.052 NA 0.064 0.0326 0.0617 ...
#>  $ L1      : num  0.1041 NA 0.0915 0.0483 0.0871 ...
#>  $ L2      : num  0.0888 NA 0.1392 0.1137 0.223 ...
#>  $ P       : num  0.0469 NA 0.1192 0.2159 0.2816 ...
#>  $ S       : num  4.91e-02 NA 2.79e-02 6.94e-18 1.98e-02 ...
#>  $ Nyc1    : num  0.132 NA 0.1981 0.2298 0.0777 ...
#>  $ Nyc2    : num  0.412 NA 0.184 0.291 0.084 ...
#>  $ Nyc3    : num  0.0669 NA 0.0596 0.0142 0.0625 ...
#>  $ Nspectra: num  41 0 237 2 78 661 17 1690 98 55 ...
#>  $ Npep    : num  8 0 41 2 14 29 6 62 11 22 ...
```

```
protNSA_2 <-
          profileSummarize(protsCombineCnew=flagSpectraPeps,
          numRefCols=7, numDataCols=9, eps=eps, GroupBy="protId",
          outlierExclude="spectra", cpus=4)
str(protNSA_2,  strict.width="cut", width=65)
```

```
#> 'data.frame':    40 obs. of  12 variables:
#>  $ prot    : chr  "ACP2" "AIF1" "ATP1A1" "B4GALT1" ...
#>  $ N       : num  0.0483 0.0858 0.1175 0.0542 0.1018 ...
#>  $ M       : num  0.052 0.037 0.0643 0.0326 0.063 ...
#>  $ L1      : num  0.1041 0.0746 0.0927 0.0483 0.0882 ...
#>  $ L2      : num  0.0888 0.0763 0.1402 0.1137 0.2211 ...
#>  $ P       : num  0.0469 0.0609 0.1192 0.2159 0.2768 ...
#>  $ S       : num  4.91e-02 1.02e-01 2.80e-02 6.94e-18 2.24e-02 ..
#>  $ Nyc1    : num  0.132 0.3907 0.1979 0.2298 0.0791 ...
#>  $ Nyc2    : num  0.4118 0.1406 0.1801 0.2914 0.0841 ...
#>  $ Nyc3    : num  0.0669 0.0325 0.0602 0.0142 0.0636 ...
#>  $ Nspectra: num  41 10 240 2 80 ...
#>  $ Npep    : num  8 5 43 2 16 30 7 71 12 27 ...
```

```
protNSA_3 <-
          profileSummarize(protsCombineCnew=flagSpectraPeps,
          numRefCols=7, numDataCols=9, eps=eps, GroupBy="protId",
          outlierExclude="none", cpus=4)
str(protNSA_3,  strict.width="cut", width=65)
```

```
#> 'data.frame':    40 obs. of  12 variables:
#>  $ prot    : chr  "ACP2" "AIF1" "ATP1A1" "B4GALT1" ...
#>  $ N       : num  0.0472 0.0858 0.1197 0.0542 0.1022 ...
#>  $ M       : num  0.0513 0.037 0.0644 0.0326 0.0628 ...
#>  $ L1      : num  0.1052 0.0746 0.094 0.0483 0.0884 ...
#>  $ L2      : num  0.0892 0.0763 0.1393 0.1137 0.2215 ...
#>  $ P       : num  0.0458 0.0609 0.1179 0.2159 0.2735 ...
#>  $ S       : num  4.78e-02 1.02e-01 2.86e-02 6.94e-18 2.28e-02 ..
#>  $ Nyc1    : num  0.1264 0.3907 0.1962 0.2298 0.0785 ...
#>  $ Nyc2    : num  0.4196 0.1406 0.1784 0.2914 0.0855 ...
#>  $ Nyc3    : num  0.0674 0.0325 0.0614 0.0142 0.0649 ...
#>  $ Nspectra: num  42 10 258 2 92 ...
#>  $ Npep    : num  8 5 43 2 16 30 7 71 12 27 ...
```

When comparing the total numbers of peptides and spectra used to calculate protein profiles, we see that protNSA_3 > protNSA_2 > protNSA_1 for each:

```
sum((protNSA_3$Nspectra))
```

```
#> [1] 6696
```

```
sum((protNSA_2$Nspectra))
```

```
#> [1] 6253
```

```
sum((protNSA_1$Nspectra))
```

```
#> [1] 6033
```

```
sum((protNSA_3$Npep))
```

```
#> [1] 944
```

```
sum((protNSA_2$Npep))
```

```
#> [1] 944
```

```
sum((protNSA_1$Npep))
```

```
#> [1] 857
```

We can examine the number of proteins with missing profile values (caused by outlier rejection) in each of the three cases by choosing one fraction, e.g., "M"; all other channels will also be missing.

```
sum(is.na(protNSA_1$M))
```

```
#> [1] 1
```

```
sum(is.na(protNSA_2$M))
```

```
#> [1] 0
```

```
sum(is.na(protNSA_3$M))
```

```
#> [1] 0
```

We see that none are rejected in this test data frame. It is important to note that the outlier rejection process includes a random component, so that repeating the analysis on the embedded data set without specifically setting the seed for the random number generator could potentially result in a different number of proteins with missing profiles.

Taking the first option (rejecting outlier spectra within peptides as well as outlier peptides) as the preferred method, we then create a protein profile data frame in the same format as used in earlier tutorials by moving the protein names from a variable within the data frame to row names.

```
protNSA_new <- protNSA_1
#copy first field in data frame containing protein names to rownames field
rownames(protNSA_new) <- protNSA_1[,1]
protNSA_new <- protNSA_new[,-1]  #delete first internal column containing protein names
```

The data frame `protNSA_new` is essentially identical to the data set `protNSA_AT5tmtMS2` included in the `protlocassign` package. (The argument `countEQ=TRUE` causes the function to compute the mean error across all profiles, not just among those that differ.)

```
data(protNSA_test)
all.equal(protNSA_test, protNSA_new, countEQ=TRUE, tolerance=0)
```

```
#> [1] "Component \"N\": Mean absolute difference: 8.074905e-12"    "Component \"M\": Mean absolute dif
#> [3] "Component \"L1\": Mean absolute difference: 1.899774e-11"   "Component \"L2\": Mean absolute di
#> [5] "Component \"P\": Mean absolute difference: 2.24403e-11"     "Component \"S\": Mean absolute dif
#> [7] "Component \"Nyc1\": Mean absolute difference: 2.600165e-11" "Component \"Nyc2\": Mean absolute
#> [9] "Component \"Nyc3\": Mean absolute difference: 9.573123e-12"
```

Creating a combined protein/peptide profile data frame

In this section, we describe a function `protPepProfile` that creates a data frame with each protein profile followed by the peptide profiles associated with each protein and their outlier status. The function requires two arguments. The first is a data frame of peptide profiles specified by `flagPeps = flagPepsBox`. The second is a data frame of protein profiles specified by `protProfileData = protNSA_1`; this data frame must have `prot` in the first column and the profiles in the following `numDataCols` columns. These two data frames must be in a consistent form and use the same type of profile data (e.g., NSA), as created earlier in this tutorial. We first re-examine the list of column names of `flagPepsBox` to determine that there are four reference columns preceding the nine columns of profile data:

```
data.frame(names(flagPepsBox))
```

```
#>                 names.flagPepsBox.
#> 1                             prot
#> 2                          peptide
#> 3  PeptidesStartPositionInProtein
#> 4             outlier.num.peptides
#> 5                                N
#> 6                                M
#> 7                               L1
#> 8                               L2
#> 9                                P
#> 10                               S
#> 11                            Nyc1
#> 12                            Nyc2
#> 13                            Nyc3
#> 14                        Nspectra
#> 15                            Npep
#> 16                          protId
#> 17                           pepId
```

```
protPepProfileNSA <- protPepProfile(flagPeps=flagPepsBox,
                                    numRefCols=4, numDataCols=9,
                                    protProfileData=protNSA_1)
str(protPepProfileNSA, strict.width="cut", width=65)
```

```
#> 'data.frame':    984 obs. of  15 variables:
#>  $ prot                          : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                       : chr  "ACP2" "ACP2::CPLQDFLR"..
#>  $ PeptidesStartPositionInProtein: int  NA 349 54 71 357 406 26..
#>  $ outlier.num.peptides          : num  NA 0 0 0 0 0 0 0 0 NA ...
#>  $ N                             : num  0.0483 0.0407 0.0439 0...
#>  $ M                             : num  0.052 0.0528 0.0413 0.0..
#>  $ L1                            : num  0.1041 0.1232 0.1074 0...
#>  $ L2                            : num  0.0888 0.0884 0.0784 0...
#>  $ P                             : num  0.0469 0.0303 0.0371 0...
#>  $ S                             : num  0.0491 0.0216 0.0559 0...
#>  $ Nyc1                          : num  0.132 0.125 0.119 0.138..
#>  $ Nyc2                          : num  0.412 0.451 0.458 0.325..
#>  $ Nyc3                          : num  0.0669 0.0671 0.0589 0...
#>  $ Nspectra                      : num  41 3 4 14 4 6 1 7 2 0 ...
#>  $ Npep                          : num  8 1 1 1 1 1 1 1 1 0 ...
```

Finally, for later CPA analysis, we need the data frame to contain a unique row name for each profile, which we obtain from the peptide column. Here, for peptides this is a concatenation of the protein name and peptide sequence while for proteins this is the protein name alone.

```
rownames(protPepProfileNSA) <- protPepProfileNSA$peptide
```

We examine the first few row names here:

```
head(rownames(protPepProfileNSA))
```

```
#> [1] "ACP2"                  "ACP2::CPLQDFLR"       "ACP2::DPYQEEK"        "ACP2::EGMLQHWELGQALR" "ACP
```

We may use `protPepProfileNSA` in carrying out transformations and in calculating CPA estimates with the `fitCPA` function. These calculations, as well as transformations, are discussed in the next section.

RSA transformations

In this section we shall first calculate RSA transformed profiles for all proteins and peptides, and then compute the constrained proportional assignments (CPA) for all proteins and peptides in a form ready for export. Then we show how to use it to plot profiles for any protein and its component peptides, with outlier peptides labelled in the plot.

First, we attach the `protlocassign` package, which includes the `protPepProfile` data frame. Note that for rows containing proteins, the peptide column contains just the protein name while for rows containing peptides, the peptide column contains a concatenated protein and peptide sequence. As in previous tutorials, we rename the embedded data frames to remove experiment specific designations (e.g., AT5tmtMS2) for ease of presentation.

```
library(protlocassign)

data(totProtAT5)
protPepNSA <- protPepProfileNSA
str(protPepNSA, strict.width="cut", width=65)
```

```
#> 'data.frame':    984 obs. of  15 variables:
#>  $ prot                          : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                       : chr  "ACP2" "ACP2::CPLQDFLR"..
#>  $ PeptidesStartPositionInProtein: int  NA 349 54 71 357 406 26..
#>  $ outlier.num.peptides          : num  NA 0 0 0 0 0 0 0 0 NA ...
#>  $ N                             : num  0.0483 0.0407 0.0439 0...
#>  $ M                             : num  0.052 0.0528 0.0413 0.0..
#>  $ L1                            : num  0.1041 0.1232 0.1074 0...
#>  $ L2                            : num  0.0888 0.0884 0.0784 0...
#>  $ P                             : num  0.0469 0.0303 0.0371 0...
#>  $ S                             : num  0.0491 0.0216 0.0559 0...
#>  $ Nyc1                          : num  0.132 0.125 0.119 0.138..
#>  $ Nyc2                          : num  0.412 0.451 0.458 0.325..
#>  $ Nyc3                          : num  0.0669 0.0671 0.0589 0...
#>  $ Nspectra                      : num  41 3 4 14 4 6 1 7 2 0 ...
#>  $ Npep                          : num  8 1 1 1 1 1 1 1 1 0 ...
```

```
totProt <- totProtAT5
totProt
```

```
#>          N          M         L1         L2          P          S      Nyc.1      Nyc.2      Nyc.3
#> 46.0447760 48.9559540  1.3840830  1.5663240 24.0455840 58.1818180  0.0368564  0.0684596  1.2730159
```

Next, we extract the NSA reference profiles from the nine profile columns of `protPepNSA`:

```
data(markerListJadot)
refLocationProfilesNSA <- locationProfileSetup(profile=protPepNSA[, 4 + (1:9)],
                          markerList=markerListJadot, numDataCols=9)
round(refLocationProfilesNSA, digits=4)
```

```
#>            N      M     L1     L2      P      S   Nyc1   Nyc2   Nyc3
#> Cyto  0.0751 0.0627 0.0717 0.0708 0.0501 0.3935 0.1229 0.1020 0.0513
#> ER    0.0938 0.0680 0.0973 0.2142 0.2456 0.0347 0.0835 0.0863 0.0765
#> Golgi 0.0860 0.0582 0.0584 0.0921 0.3114 0.0288 0.1692 0.1623 0.0337
#> Lyso  0.0419 0.0500 0.1036 0.0740 0.0411 0.0449 0.1431 0.4375 0.0640
#> Mito  0.1297 0.2881 0.1554 0.0734 0.0432 0.0463 0.0767 0.0675 0.1197
#> Nuc   0.7406 0.0306 0.0150 0.0341 0.0370 0.0405 0.0365 0.0380 0.0275
#> Perox 0.0525 0.0694 0.2821 0.1382 0.0383 0.0522 0.0664 0.0833 0.2176
#> PM    0.1165 0.0641 0.0820 0.1185 0.1126 0.0371 0.2380 0.1791 0.0521
```

Using the `RSAfromNSA` function described previously in Tutorial 3, we calculate the RSA-transformed marker profiles:

```
refLocationProfilesRSA <- RSAfromNSA(NSA=refLocationProfilesNSA,
                              NstartMaterialFractions=6, totProt=totProtAT5)
round(refLocationProfilesRSA, digits=4)
```

```
#>            N      M     L1     L2      P      S   Nyc1   Nyc2   Nyc3
#> Cyto  0.4388 0.3664 0.4190 0.4135 0.2925 2.2992 0.7183 0.5959 0.2997
#> ER    1.0531 0.7638 1.0928 2.4060 2.7583 0.3899 0.9381 0.9691 0.8590
#> Golgi 0.9569 0.6472 0.6497 1.0243 3.4638 0.3204 1.8824 1.8056 0.3748
#> Lyso  0.9160 1.0937 2.2656 1.6188 0.8988 0.9826 3.1310 9.5709 1.3991
#> Mito  0.9683 2.1502 1.1599 0.5481 0.3221 0.3458 0.5726 0.5038 0.8933
#> Nuc   3.4282 0.1418 0.0694 0.1580 0.1715 0.1877 0.1690 0.1761 0.1272
#> Perox 0.9114 1.2046 4.8972 2.3997 0.6655 0.9058 1.1534 1.4464 3.7779
#> PM    1.5361 0.8449 1.0806 1.5617 1.4850 0.4887 3.1374 2.3616 0.6870
```

We transform the protein/peptide profiles by taking the nine columns containing the profile data from `protPepNSA` and then, using the `RSAfromNSA` function described previously in Tutorial 3, we calculate an intermediate nine-column data frame `protPepRSA_trimmed` of RSA-transformed profiles.

```
protPepRSA_trimmed <- RSAfromNSA(NSA=protPepNSA[, 4 + (1:9)],
                              NstartMaterialFractions=6, totProt=totProtAT5)
str(protPepRSA_trimmed, strict.width="cut", width=65)
```

```
#> 'data.frame':    984 obs. of  9 variables:
#>  $ N   : num  0.963 1.085 0.935 0.939 0.935 ...
#>  $ M   : num  1.036 1.41 0.879 0.963 1.512 ...
#>  $ L1  : num  2.07 3.29 2.29 1.41 3.08 ...
#>  $ L2  : num  1.77 2.36 1.67 1.51 2.29 ...
#>  $ P   : num  0.935 0.808 0.79 0.991 0.776 ...
#>  $ S   : num  0.98 0.575 1.191 1.059 0.629 ...
#>  $ Nyc1: num  2.63 3.35 2.53 2.02 3.38 ...
#>  $ Nyc2: num  8.21 12.03 9.76 4.76 11.7 ...
#>  $ Nyc3: num  1.33 1.79 1.25 0.99 1.68 ...
```

Finally, we add the five reference columns back in as the first columns of `protPepRSA` and also the two columns listing the number of spectra and peptides per protein. The resulting data frame `protPepRSA` has the same structure as the original data frame `protPepNSA`.

```
protPepRSA <- data.frame(protPepNSA[, 1:4], protPepRSA_trimmed, protPepNSA[,14:15] )  # add in the ref
str(protPepRSA, strict.width="cut", width=65)
```

```
#> 'data.frame':    984 obs. of  15 variables:
#>  $ prot                          : chr  "ACP2" "ACP2" "ACP2" ""..
#>  $ peptide                       : chr  "ACP2" "ACP2::CPLQDFLR"..
#>  $ PeptidesStartPositionInProtein: int  NA 349 54 71 357 406 26..
#>  $ outlier.num.peptides          : num  NA 0 0 0 0 0 0 0 0 NA ...
#>  $ N                             : num  0.963 1.085 0.935 0.939..
#>  $ M                             : num  1.036 1.41 0.879 0.963 ..
#>  $ L1                            : num  2.07 3.29 2.29 1.41 3.0..
#>  $ L2                            : num  1.77 2.36 1.67 1.51 2.2..
#>  $ P                             : num  0.935 0.808 0.79 0.991 ..
#>  $ S                             : num  0.98 0.575 1.191 1.059 ..
#>  $ Nyc1                          : num  2.63 3.35 2.53 2.02 3.3..
#>  $ Nyc2                          : num  8.21 12.03 9.76 4.76 11..
#>  $ Nyc3                          : num  1.33 1.79 1.25 0.99 1.6..
#>  $ Nspectra                      : num  41 3 4 14 4 6 1 7 2 0 ...
#>  $ Npep                          : num  8 1 1 1 1 1 1 1 1 0 ...
```

Plotting RSA protein and peptide profiles

Next, we identify rows with proteins only, and extract them. The resulting data frame, `protRSA`, parallels the structure of `protNSA`. We also extract the rows with peptides only in the data frame `pepRSA`.

```
protRSA.ind <- {protPepRSA$prot == protPepRSA$peptide}  # protein indicators
protRSA <- protPepRSA[protRSA.ind,]  # these are the data for proteins only
dim(protRSA)
```

```
#> [1] 40 15
```

```
pepRSA <- protPepRSA[!protRSA.ind,] # these are the data for peptides only

data.frame(colnames(protRSA))
```

```
#>                 colnames.protRSA.
#> 1                            prot
#> 2                         peptide
#> 3  PeptidesStartPositionInProtein
#> 4            outlier.num.peptides
#> 5                               N
#> 6                               M
#> 7                              L1
#> 8                              L2
#> 9                               P
#> 10                              S
#> 11                          Nyc1
```

```
#> 12                          Nyc2
#> 13                          Nyc3
#> 14                      Nspectra
#> 15                          Npep
```

Now we calculate the constrained proportional assignments on proteins only, using RSA-transformed profiles:

```
protCPAfromRSA <-  fitCPA(profile=protRSA[, 4+1:9],
                      refLocationProfiles=refLocationProfilesRSA,
                      numDataCols=9)
```

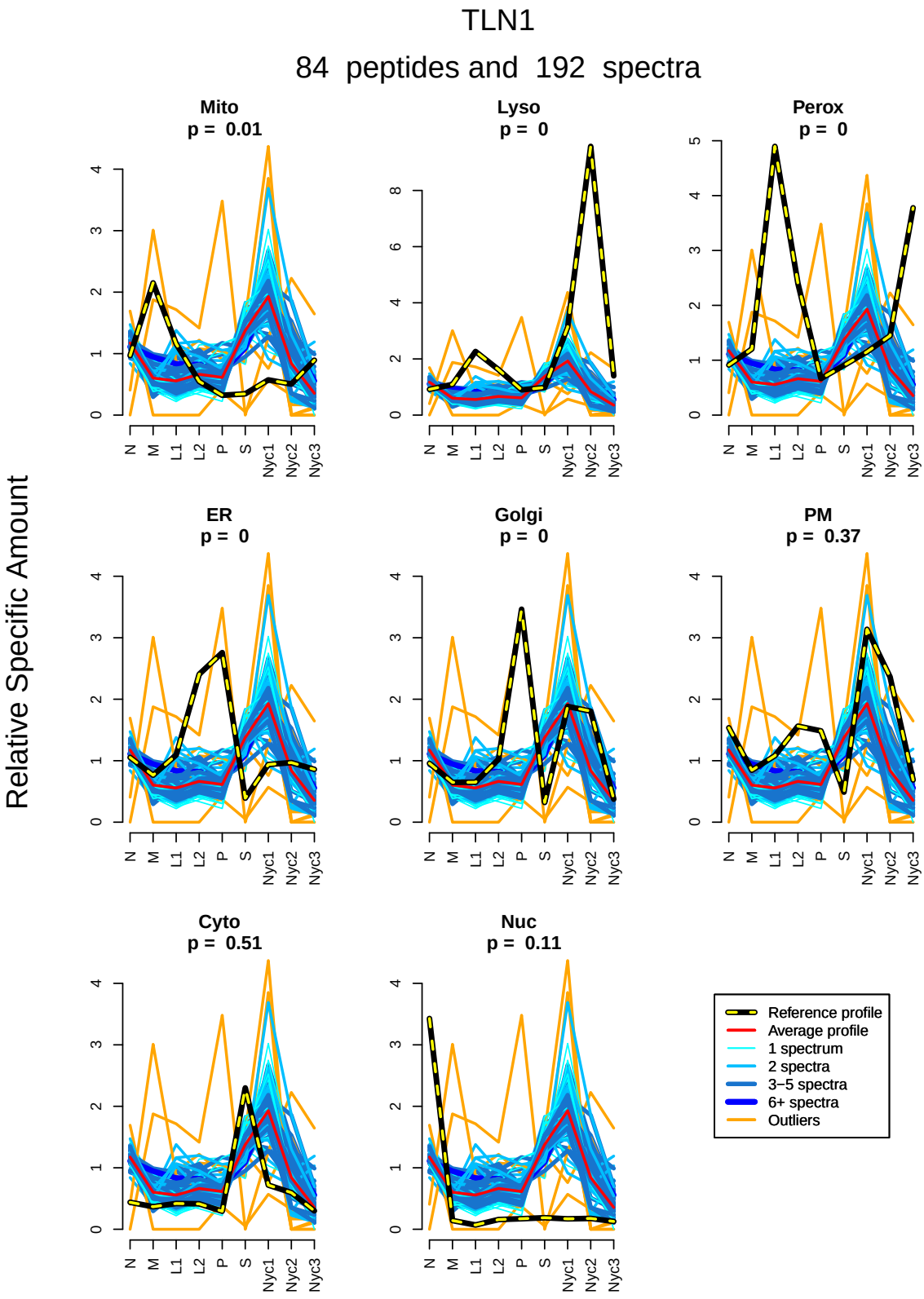
```
#> cpa does not converge for a protein
#>  returning missing values for cpa estimates for that protein
```

```
str(protCPAfromRSA, strict.width="cut", width=65)
```

```
#> 'data.frame':    40 obs. of  8 variables:
#>  $ Cyto : num  0.0341 NA 0 0 0 ...
#>  $ ER   : num  0.0842 NA 0.1738 0 0.9537 ...
#>  $ Golgi: num  0 NA 0 0.5756 0.0463 ...
#>  $ Lyso : num  0.8399 NA 0.0553 0.4244 0 ...
#>  $ Mito : num  0 NA 0 0 0 ...
#>  $ Nuc  : num  0.0207 NA 0.0165 0 0 ...
#>  $ Perox: num  0.0211 NA 0.0232 0 0 ...
#>  $ PM   : num  0 NA 0.731 0 0 ...
```

The following commands generate a plot of TLN1 protein/peptides, with CPA estimates. Outlier peptide profiles are in orange. The header reports the number of peptides and spectra used to compute the protein profile, which in this case excludes outlier peptides and outlier spectra.

```
#windows(width=7.5, height=10)  # open a window 7.5 by 10 inches
protPepPlotfun(protName="TLN1", protProfile=protRSA[,5:15],
               Nspectra=TRUE, pepProfile=pepRSA, numRefCols=4,
               numDataCols=9, n.compartments=8,
               refLocationProfiles=refLocationProfilesRSA,
               assignPropsMat=protCPAfromRSA,
               yAxisLabel="Relative Specific Amount")
```



Note that the outlier peptides do not contribute to the CPA analysis of the proteins, but these may be of

interest. For instance, they may represent protein isoforms with distinct distributions. Thus, there may be specific biological questions that require CPA estimates for all proteins and peptides without outlier removal. This can be accomplished using the following command:

```
protPepCPAfromRSA <- fitCPA(profile=protPepRSA[,4 + 1:9],
                            refLocationProfiles=refLocationProfilesRSA, numDataCols=9)
```

```
#> cpa does not converge for a protein
#>  returning missing values for cpa estimates for that protein
#> [1] "500 profiles fit"
```

```
str(protPepCPAfromRSA, strict.width="cut", width=65)
```

```
#> 'data.frame':    984 obs. of  8 variables:
#>  $ Cyto : num  0.03409 0 0.00314 0.20691 0 ...
#>  $ ER   : num  0.0842 0 0 0.1624 0 ...
#>  $ Golgi: num  0 0 0 0 0 ...
#>  $ Lyso : num  0.84 1 0.997 0.44 1 ...
#>  $ Mito : num  0 0 0 0.0838 0 ...
#>  $ Nuc  : num  0.0207 0 0 0.012 0 ...
#>  $ Perox: num  0.02114 0 0 0.00337 0 ...
#>  $ PM   : num  0 0 0 0.0919 0 ...
```

We next assemble the final CPA values for the protein/peptide data along with ancillary information, ready for export. Then we output the data to C:\temp\myProteinOutput; users will select their own directory.

```
protPepCPAfromRSAout <- data.frame(protPepRSA[,1:4], protPepCPAfromRSA, protPepRSA[,14:15])
protPepCPAfromRSAout$prot <- paste("'", protPepCPAfromRSAout$prot, sep="")
protPepCPAfromRSAout$peptide <- paste("'", protPepCPAfromRSAout$peptide, sep="")

setwd("C:\\temp\\myProteinOutput")
write.csv(protPepCPAfromRSAout, file="protPepCPAfromRSAout.csv", row.names=FALSE, na=".")
```

To output plots of all of the protein and peptide profiles into a single pdf file, we first use `setwd` to point to the desired output directory, and then we can set up a loop as follows:

```
setwd("C:\\temp\\myProteinOutput")
pdf(file="allProtPepPlotsRSA.pdf", width=7, height=10)
n.prots <- nrow(protRSA)
for (i in 1:n.prots) {
   protPepPlotfun(protName=protRSA$prot[i],
       protProfile=protRSA[,5:15],
       Nspectra=TRUE, pepProfile=pepRSA, numRefCols=4,
       numDataCols=9, n.compartments=8,
       refLocationProfiles=refLocationProfilesRSA,
       assignPropsMat=protCPAfromRSA,
       yAxisLabel="Relative Specific Amount")
}
dev.off()
```

References

Jadot, M.; Boonen, M.; Thirion, J.; Wang, N.; Xing, J.; Zhao, C.; Tannous, A.; Qian, M.; Zheng, H.; Everett, J. K., Accounting for protein subcellular localization: A compartmental map of the rat liver proteome. Molecular & Cellular Proteomics 2017, 16, (2), 194-212.

Tannous, A.; Boonen, M.; Zheng, H.; Zhao, C.; Germain, C. J.; Moore, D. F.; Sleat, D. E.; Jadot, M.; Lobel, P., Comparative Analysis of Quantitative Mass Spectrometric Methods for Subcellular Proteomics. J Proteome Res 2020, 19, (4), 1718-1730

Reproducibility

```
print(utils::sessionInfo(), width=80)
```

```
#> R version 4.1.3 (2022-03-10)
#> Platform: x86_64-w64-mingw32/x64 (64-bit)
#> Running under: Windows 10 x64 (build 19044)
#>
#> Matrix products: default
#>
#> locale:
#> [1] LC_COLLATE=English_United States.1252
#> [2] LC_CTYPE=English_United States.1252
#> [3] LC_MONETARY=English_United States.1252
#> [4] LC_NUMERIC=C
#> [5] LC_TIME=English_United States.1252
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] BiocParallel_1.28.3  outliers_0.14        plot.matrix_1.6.1
#> [4] pracma_2.3.8         protlocassign_0.99.1 lme4_1.1-28
#> [7] Matrix_1.4-0
#>
#> loaded via a namespace (and not attached):
#>  [1] Rcpp_1.0.8        lattice_0.20-45   snow_0.4-4        prettyunits_1.1.1
#>  [5] ps_1.6.0          rprojroot_2.0.2   digest_0.6.29     utf8_1.2.2
#>  [9] R6_2.5.1          evaluate_0.15     ggplot2_3.3.5     highr_0.9
#> [13] pillar_1.7.0      rlang_1.0.2       rstudioapi_0.13   minqa_1.2.4
#> [17] callr_3.7.0       nloptr_2.0.0      rmarkdown_2.13    desc_1.4.1
#> [21] devtools_2.4.3    splines_4.1.3     stringr_1.4.0     munsell_0.5.0
#> [25] tinytex_0.37      compiler_4.1.3    xfun_0.30         pkgconfig_2.0.3
#> [29] pkgbuild_1.3.1    htmltools_0.5.2   tibble_3.1.6      gridExtra_2.3
#> [33] BB_2019.10-1      quadprog_1.5-8    fansi_1.0.2       viridisLite_0.4.0
#> [37] crayon_1.5.0      withr_2.5.0       MASS_7.3-55       brio_1.1.3
#> [41] grid_4.1.3        nlme_3.1-155      gtable_0.3.0      lifecycle_1.0.1
#> [45] magrittr_2.0.2    scales_1.1.1      cli_3.2.0         stringi_1.7.6
#> [49] cachem_1.0.6      viridis_0.6.2     fs_1.5.2          remotes_2.4.2
#> [53] testthat_3.1.2    ellipsis_0.3.2    vctrs_0.3.8       boot_1.3-28
#> [57] tools_4.1.3       glue_1.6.2        purrr_0.3.4       processx_3.5.2
#> [61] pkgload_1.2.4     parallel_4.1.3    fastmap_1.1.0     yaml_2.3.5
```

```
#> [65] colorspace_2.0-3  sessioninfo_1.2.2 memoise_2.0.1     knitr_1.37
#> [69] usethis_2.1.5
```