Tutorial 1: Getting started with assignment of proteins to subcellular locations using protlocassign

DFM & PL

2022-03-12

Contents

Introduction

Determining the locations of proteins in the cell is an important but complex problem. A frequently employed approach for this involves centrifugation-based methods to partially separate different organelles and other cellular compartments and then to determine the relative distribution of different proteins among the centrifugation fractions. The location of proteins of interest in the cellular compartments are then inferred by comparing their distributions across the centrifugation fractions to the distributions of a set of reference proteins (markers) with known cellular locations.

This package implements a subcellular protein assignment procedure known as "constrained proportional assignment", or CPA (Jadot et al, 2016). The basic concepts involved in CPA and analysis of subcellular proteomics data are described in the main paper. These tutorials serve as a basis for implementing CPA and various utilities for both experienced and beginner R users.

There are two fundamental inputs for CPA. The first is a file with information regarding the distribution of each protein or other species of interest to be analyzed across centrifugation fractions. Each row has a name that serves as a unique identifier, which we here refer to as a protein name, but one can use other identifiers, (e.g., protein group, protein isoform, gene name, accession number, etc.). For each protein (or other identifier), there are data that reflects its distribution among the centrifugal fractions, which represent a profile. The second is a file containing a list of single-compartment reference proteins (markers) and their associated subcellular locations. As an initial step, the package uses the markers to compute profiles for individual cellular compartments. Then, for each protein, it finds the best match for its profile using a linear combination of compartment profiles. The relative weights of the linear combination in principle reflect the relative abundance of a given protein among different compartments. The CPA method can thus account for proteins that have multiple residences, and estimate the relative proportion among these residences.

We illustrate with an example. Tannous et al. (2020) presented an experiment (designated here as AT5) analyzing abundance levels of proteins across a total of nine fractions: six fractions (N, M, L1, L2, P, and S) from differential centrifugation of a rat liver homogenate and three fractions (Nyc1, Nyc2, and Nyc3) from a Nycodenz density gradient centrifugation of the differential fraction L1. Eight subcellular compartments were considered: nucleus (Nuc), mitochondria (Mito), lysosomes (Lyso), peroxisomes (Perox), endoplasmic reticulum (ER) Golgi apparatus (Golgi), plasma membrane (PM), and cytosol (Cyto). The CPA method assigns each of a large number of proteins to one or more of these compartments, based on profiles from the set of reference proteins.

Installing the protlocassign package

The `protlocassign` package is located on the Bioconductor respository. The first step is to install the "BiocManager" package from CRAN, by typing:

```
install.packages("BiocManager")
```

Then install the protlocassign package from the github repository by typing:

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

This will make the programs and data sets available. Also, several libraries are required which may be downloaded from the CRAN respository by typing:

```
BiocManager::install(c("BB", "pracma", "lme4", "outliers"))
```

Finally, the parallel processing package `BiocParallel` from the Bioconductor package is needed to use parallel processing. The following code will install the package if it is not already present:

```
BiocManager::install("BiocParallel")
```

Once you have installed these packages, you do not need to re-install them the next time you start up R.

Working with data

To use this package, you will need two data sets. One, the protein profile data set, contains rows with a unique identifier followed by data describing the profile associated with each identifier across a series of centrifugation fractions. In this tutorial, the profiles associated with each identifier are specific amounts with sums constrained to 1 ("normalized specific amounts" or NSAs, see main Paper) but can be in a different form or further transformed to improve the quality of the fit (see Tutorials 2-4). The protein profile data set may contain two additional values representing the numbers of peptides and spectra that were used to compute the profile (see below). The other data set consists of the list of reference proteins and their associated known subcellular compartments.

Consider for example the TMT MS2 data from Tannous et al. (2020) experiment AT5. To get started, load the package and attach the embedded protein profile data set; we see that it has 7893 rows and 11 columns:

```
library(protlocassign)
data(protNSA_AT5tmtMS2)
dim(protNSA_AT5tmtMS2)
```

```
#> [1] 7894   11
```

Alternatively we may work with a test data set containing a small subset of the proteins

```
data(protNSA_test)
dim(protNSA_test)
```

```
#> [1] 40 11
```

For the sake of brevity, we here use the `protNSA_test`data frame and rename it `protNSA`:

```
protNSA <- protNSA_test
```

The first few rows of the data can be examined using the `head` command, rounding to improve legibility. The first nine columns represent the protein profile. The last two columns, which are optional, give the number of spectra and the number of peptides (sequences) for each protein. Note that while we use a nested random effects model described in Jadot et al., 2017 to compute the means across spectra (also see Tutorial 5), other methods, including taking a straight average or weighted average (e.g., based on reporter ion intensities or peak areas), may be appropriate for other applications.

```
round(head(protNSA), digits=2)
```

```
#>            N    M   L1   L2    P    S Nyc1 Nyc2 Nyc3 Nspectra Npep
#> ACP2    0.05 0.05 0.10 0.09 0.05 0.05 0.13 0.41 0.07       41    8
#> AIF1      NA   NA   NA   NA   NA   NA   NA   NA   NA        0    0
#> ATP1A1  0.12 0.06 0.09 0.14 0.12 0.03 0.20 0.18 0.06      237   41
#> B4GALT1 0.05 0.03 0.05 0.11 0.22 0.00 0.23 0.29 0.01        2    2
#> CANX    0.10 0.06 0.09 0.22 0.28 0.02 0.08 0.08 0.06       78   14
#> CAT     0.05 0.08 0.29 0.14 0.04 0.07 0.05 0.07 0.22      661   29
```

The list containing reference proteins is derived from Jadot et al (2016) and examining the dimensions of the data reveals that it contains 39 rows and two columns, the first for the protein names and the second for their respective subcellular compartments.

```
data(markerListJadot)
dim(markerListJadot)
```

```
#> [1] 39  2
```

The data set can be viewed by entering its name as follows:

```
markerListJadot
```

```
#>    protName referenceCompartment
#> 1      ADH1                Cyto
#> 2      DPP3                Cyto
#> 3       GPT                Cyto
#> 4      PCK1                Cyto
#> 5      PGM1                Cyto
#> 6      CANX                  ER
#> 7     GANAB                  ER
#> 8       POR                  ER
#> 9      RPN2                  ER
```

```
#> 10    UGGT1                  ER
#> 11  UGT2B37                  ER
#> 12  B4GALT1              Golgi
#> 13   MAN1A2              Golgi
#> 14   MAN2A1              Golgi
#> 15    MGAT1              Golgi
#> 16  ST6GAL1              Golgi
#> 17     ACP2               Lyso
#> 18     CTSD               Lyso
#> 19     GLB1               Lyso
#> 20     HEXA               Lyso
#> 21   Mt-CO2               Mito
#> 22     CPS1               Mito
#> 23    GLUD1               Mito
#> 24     MDH2               Mito
#> 25       PC               Mito
#> 26     SDHA               Mito
#> 27 HIST1H1D                Nuc
#> 28     LMNA                Nuc
#> 29    LMNB1                Nuc
#> 30    LMNB2                Nuc
#> 31      CAT              Perox
#> 32     HAO1              Perox
#> 33     PHYH              Perox
#> 34      UOX              Perox
#> 35   ATP1A1                 PM
#> 36     CD38                 PM
#> 37    ENPP1                 PM
#> 38    ITGB1                 PM
#> 39     NT5E                 PM
```

While the `protNSA` and `markerListJadot` data sets are included in the package, they were initially read into R from external files and automatically converted to data frames. This will need to be done for any new data set. As an example, we demonstrate this for a case where one is working with the Windows operating system and the data sets reside in the directory `C:\temp\myproteindata`. (If one is working with either the Linux or Mac OS, appropriate changes to these procedures will be needed to access files.) First, set the working directory to point there, either by navigating to it in R studio (session menu pane) or by entering:

```
setwd("C:\\temp\\myproteindata")
```

Note that in R each backslash character must be doubled to be interpreted correctly (since otherwise it will be incorrectly interpreted as an escape character). Alternatively, you may use a single forward slash:

```
setwd("C:/temp/myproteindata")
```

To illustrate how to read in the protein profiles and list of marker proteins, we write out our protein profiles and marker list as comma-delimited files. For the protein profiles, we write out the protein names (which are row names in R) by specifying `row.names=T`. Note that we first alter the protein names so that they are preceded by a single quote since, if one subsequently opens the csv file with excel, it could automatically reformat some names to dates (e.g., March1 to 1-Mar). To write protein profiles and marker proteins to `.csv` files, input:

```
protNSAout <- protNSA
```

```
rownames(protNSAout) <- paste("'", rownames(protNSA), sep="")
markerListJadotOut <- markerListJadot
markerListJadotOut$protName <- paste("'", markerListJadot$protName, sep="")
write.csv(protNSAout, file="protNSAout.csv", row.names=T)
write.csv(markerListJadotOut, file="markerListJadotOut.csv", row.names=F)
```

We may examine these files by importing them into Excel. For the marker protein data file, the first row must contain the two column names (protein unique identifier and compartment designation). Note that the marker proteins must be specified precisely as listed in the protein profile data set.

We then read in the two data sets (protein profile data and reference protein list), which must be in comma-separated format (`.csv`), with the first row containing column names. The option `row.names=1` takes the first column of the `protNSAout.csv` file and uses it as row names for the R file `MyPProtNSAin`.

```
MyProtNSAin <- read.csv(file="protNSAout.csv", row.names=1)
MyMarkerListIn <- read.csv(file="markerListJadotOut.csv")
```

We then remove the preceding single quotes from the protein names.

```
rownames(MyProtNSAin) <- sub("^'", "", rownames(MyProtNSAin))
MyMarkerListIn[,1] <- sub("^'", "", MyMarkerListIn[,1])
```

These new R data frames are identical to `protNSA` and `markerListJadot`. If the user reads in their own data, be sure that the first row in each of the ".csv" files contains column names. In particular, the names in the first row of the markers file must be "protName" and "referenceCompartment" to ensure that the resulting R data frame is in the proper format.

Creating and viewing compartment and marker protein profiles

In order to assign proteins proportionately to their respective compartments, we first use the function `locationProfileSetup` to obtain profiles for the compartments based on the means of the individual reference proteins that represent each compartment. This function produces a matrix that has one row for each compartment and one column for each fraction that comprises the profile.

```
refLocationProfilesNSA <- locationProfileSetup(profile=protNSA, markerList=markerListJadot, numDataCols=
```

We display the data, using rounding to improve readability.

```
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
```

To examine the available compartments, view the row names of `refLocationProfilesNSA`.

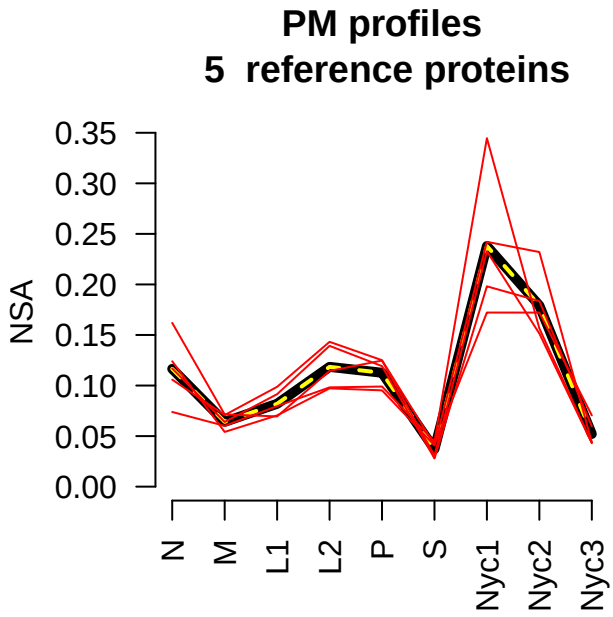
```
rownames(refLocationProfilesNSA)
```

```
#> [1] "Cyto"  "ER"    "Golgi" "Lyso"  "Mito"  "Nuc"   "Perox" "PM"
```

To graphically display a particular compartment profile and its component proteins, use `markerProfilePlot`. For example, to plot the profile for plasma membrane, input:

```
markerProfilePlot(refLoc="PM", profile=protNSA, markerList=markerListJadot,
                    refLocationProfiles=refLocationProfilesNSA, ylab="NSA")
```

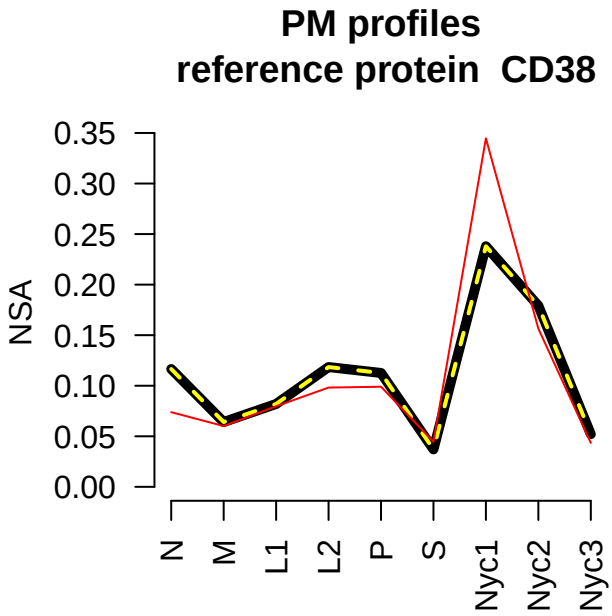
```
#> NULL
```



This displays the PM compartmental profile (dashed yellow-black line) and its five component reference proteins (red lines). To plot, for example, the second PM marker protein, use the option `refProtPlot=2` in the `markerProfilePlot` function:

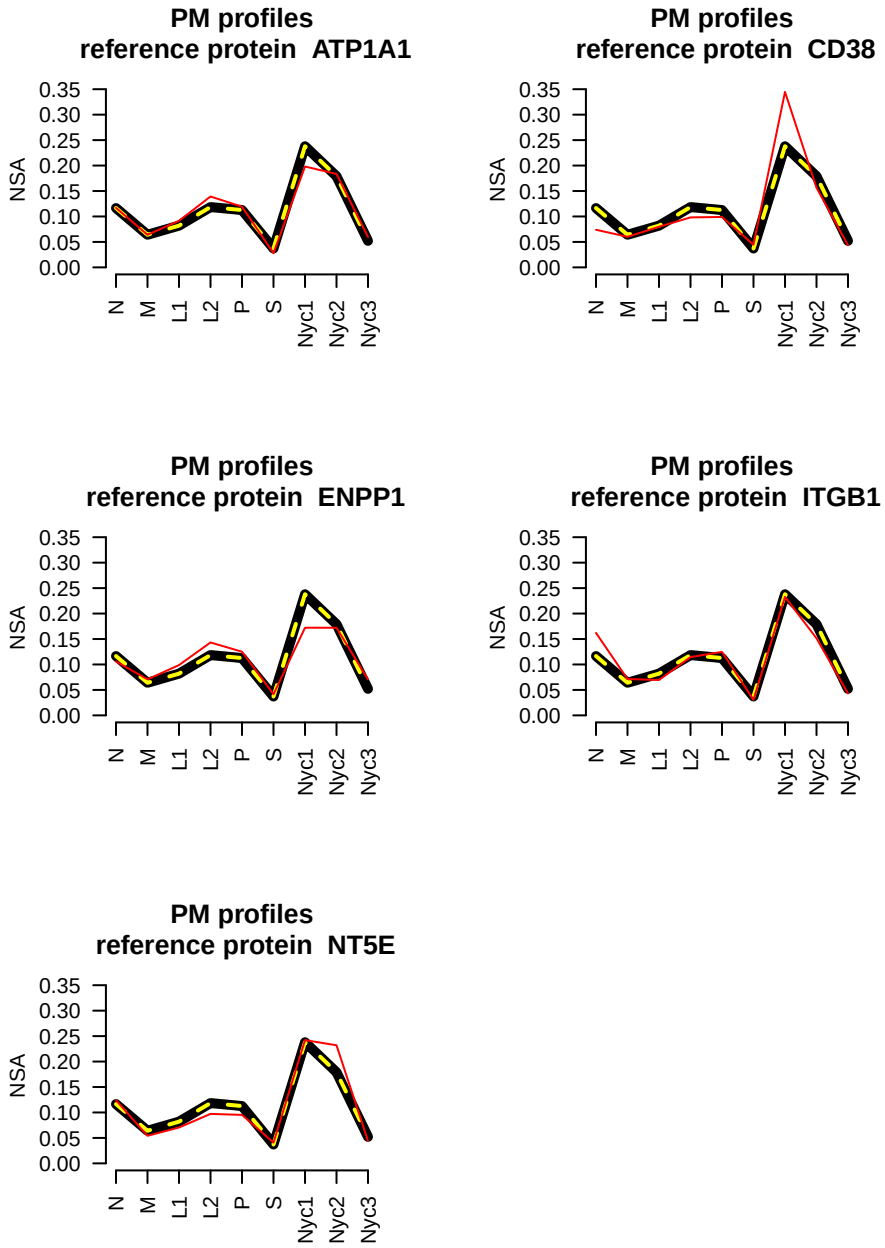
```
markerProfilePlot(refLoc="PM", profile=protNSA,
                  markerList=markerListJadot,
                  refLocationProfiles=refLocationProfilesNSA, ylab="NSA",
                  refProtPlot=2)
```

```
#> NULL
```



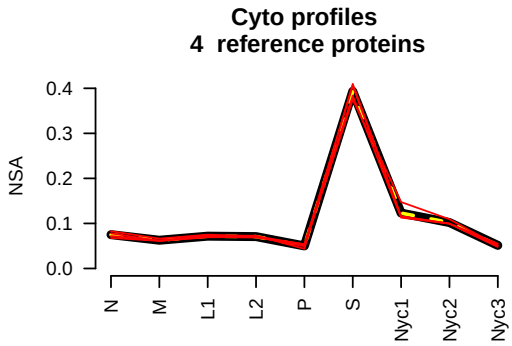
To show individually each of the five reference proteins that are used to calculate the PM compartmental profile, use `par(mfrow=c(3,2))` to set up a plot array with 3 rows and 2 columns which will accommodate up to six plots on a single page. Then plot all five of the them one-by-one by looping through the five PM marker proteins by first specifying `for (j in 1:5)` and invoking the option `refProtPlot=j` in the `markerProfilePlot` function. (To make them legible, you may first need to open a new window by entering `windows(width=5, height=7)`.) The parameters in `par(mfrow=c(3,2))` and `for (j in 1:5)` can be adjusted to display any number of plots.

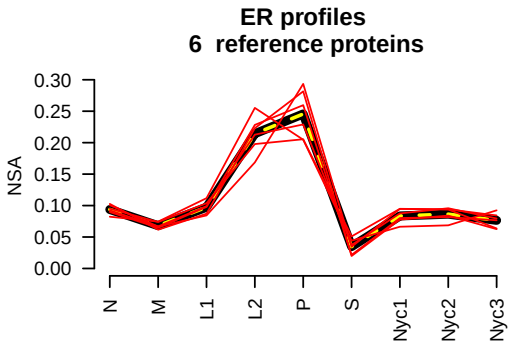
```
par(mfrow=c(3,2)) # this will be new default layout for subsequent plots.
# Will need to reset par(mfrow=c(1,1)) for single graph layouts

for (j in 1:5) {
  markerProfilePlot(refLoc="PM", profile=protNSA,
                    markerList=markerListJadot,
                    refLocationProfiles=refLocationProfilesNSA,
                    ylab="NSA", refProtPlot=j)
  }
```

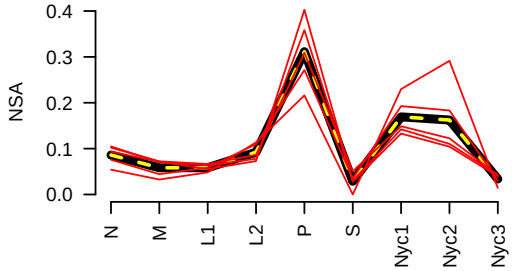


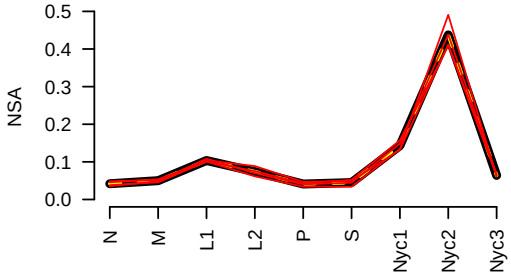
To plot all of the compartment and individual marker protein profiles, we may set up a plot window with four rows and two columns and then loop through the eight subcellular compartments:

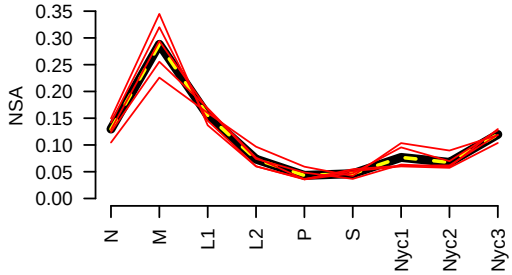
```
loc.list <- rownames(refLocationProfilesNSA)
n.loc <- length(loc.list)
par(mfrow=c(4,2))
for (i in 1:n.loc) {
  markerProfilePlot(refLoc=loc.list[i], profile=protNSA,
                    markerList=markerListJadot,
                    refLocationProfiles=refLocationProfilesNSA, ylab="NSA")
  }
```

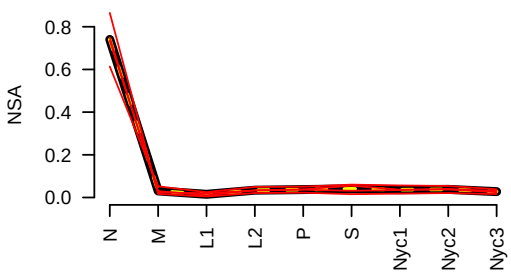

Cyto profiles
4 reference proteins
NSA
0.0
0.1
0.2
0.3
0.4
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3

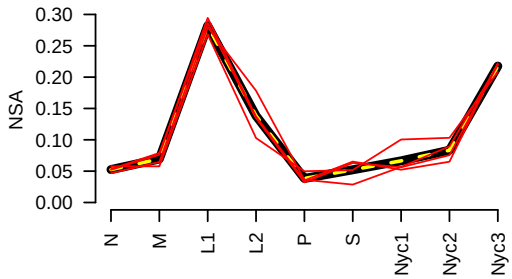

ER profiles
6 reference proteins
NSA
0.00
0.05
0.10
0.15
0.20
0.25
0.30
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3

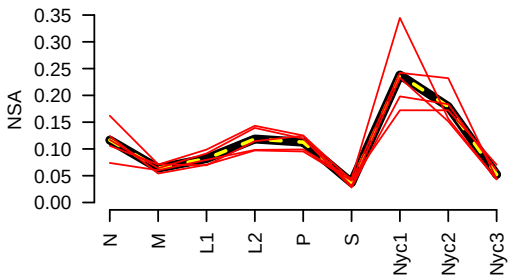

Golgi profiles
5 reference proteins
NSA
0.0
0.1
0.2
0.3
0.4
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Lyso profiles
4 reference proteins
NSA
0.0
0.1
0.2
0.3
0.4
0.5
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Mito profiles
5 reference proteins
NSA
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Nuc profiles
4 reference proteins
NSA
0.0
0.2
0.4
0.6
0.8
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Perox profiles
4 reference proteins
NSA
0.00
0.05
0.10
0.15
0.20
0.25
0.30
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


PM profiles
5 reference proteins
NSA
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3

Obtaining constrained proportionate assignments of proteins to compartments

Once we are satisfied with the marker set and compartment profiles, we can run the CPA routine which, for each protein in our data set, apportions its residency among the different specified compartments. This may take several minutes to complete. Note that the command below uses default options. It is also possible to specify optional parameters including writing out the obtained goodness of fit minimum (`minVal=T`, default is `minVal=F`), setting initial parameters for each CPA value using an eight element vector specifying the starting proportions (`startProp`, if not specified the function will assign a value of 1/8 for each of the eight compartments), and by constraining the output CPA values for specified compartments to be zero (see help file, `?fitCPA`).

```
protCPAfromNSA <- fitCPA(profile=protNSA,
                         refLocationProfiles=refLocationProfilesNSA,
                         numDataCols=9)
```

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

Note that the protein "AIF1" (protein 356) has all missing values, which is why the spg function returns an error for that one protein.

We can view the structure of the output data file and see that the row names contain the protein names, and the next 8 columns contain the allocation proportions of each protein to the eight compartments. The last two columns are integers representing the numbers of peptides and spectra, which are carried over from the input `protProfileSummary` profile data, these are not essential for the CPA analysis.

```
round(tail(protCPAfromNSA), digits=2)
```

```
#>         Cyto   ER Golgi Lyso Mito  Nuc Perox   PM Nspectra Npeptides
#> ST6GAL1 0.00 0.00  0.80 0.05 0.00 0.00  0.00 0.15       44         14
#> TLN1    0.36 0.00  0.00 0.00 0.00 0.06  0.00 0.58      192         84
#> TLN2    0.13 0.00  0.02 0.00 0.14 0.26  0.00 0.44       80         50
#> UGGT1   0.00 0.93  0.00 0.02 0.00 0.00  0.02 0.02      101         38
#> UGT2B37 0.00 1.00  0.00 0.00 0.00 0.00  0.00 0.00      143         25
#> UOX     0.00 0.03  0.00 0.00 0.00 0.00  0.97 0.00      278         14
```

We can look at the profile of, for example, the protein "TLN1". We first ensure that the protein is in the dataset:

```
protIndex("TLN1", profile=protNSA)
```

```
#>   Prot index number Prot name
#> 1                36      TLN1
```

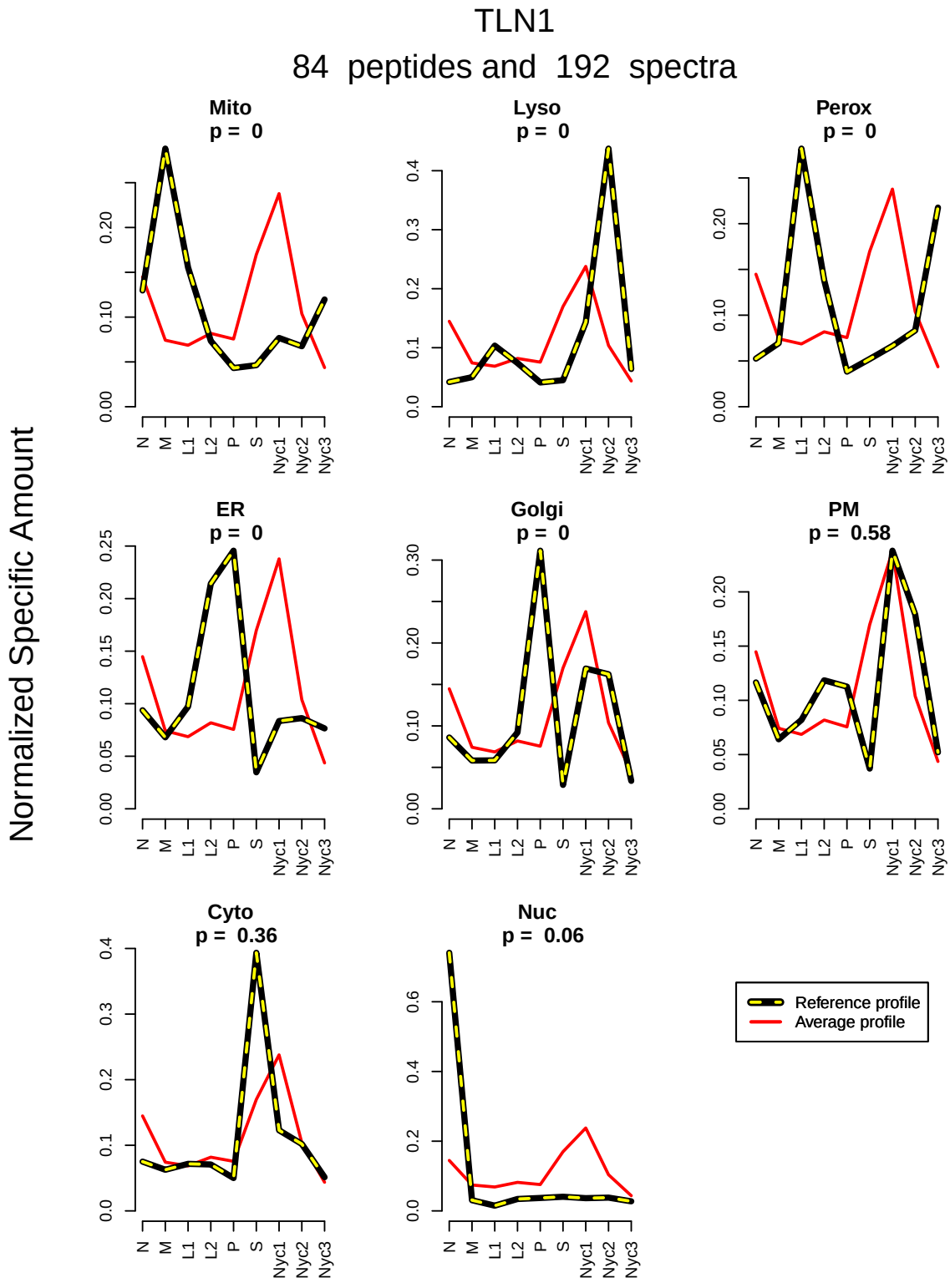
This function also accepts partial matching of the first few letters of a protein. For example, we can find the indices of the proteins starting with "TLN":

```
protIndex("TLN", profile=protNSA)
```

```
#>   Prot index number Prot name
#> 1                36      TLN1
#> 2                37      TLN2
```

Now we plot the results for protein TLN1:

```
protPlotfun(protName="TLN1", profile=protNSA,
numDataCols=9, n.compartments=8,
  refLocationProfiles=refLocationProfilesNSA,
  assignPropsMat=protCPAfromNSA,
  yAxisLabel="Normalized Specific Amount")
```



The x-axis represents the nine fractions, which are N, M, L1, L2, P, S, Nyc.1, Nyc.2, and Nyc.3. In each of the eight plots, the red line is the average profile of the protein. The dashed yellow-black lines show the expected profile for a protein entirely resident in the respective subcellular location. In this set of plots, we

see that the CPA procedure assigns a 57 percent residence proportion to plasma membrane and 36 percent residence to cytosol. The observed red profile closely matches a mixture of the yellow-black lines weighted by the indicated proportions.

The `protPlotfun` function is designed to plot profiles of eight subcellular locations. If a data set has more than eight of these, it will be necessary to modify the code to accommodate the larger number.

Saving the CPA output

Data output

To save the results of the CPA procedure, as before, we prepend the protein names with a single quote and write out a csv file to a specified directory.

```
protCPAfromNSAout <- protCPAfromNSA
rownames(protCPAfromNSAout) <- paste("'", rownames(protCPAfromNSAout), sep="")

setwd("C:/temp/myProteinOutput")

write.csv(protCPAfromNSAout, file="protCPAfromNSAout_AT5tmtMS2.csv", row.names=T)
```

Plot output as pdf files

To save the plot of a protein (`TLN1` for example) as a pdf file, we first specify a pdf plot `window`, call the `protPlotfun` function as before, and then close the plot window using `dev.off()` to allow R to complete producing the file:

```
pdf(file="myPlotPDFfile.pdf", width=7, height=10)
protPlotfun(protName="TLN1", profile=protNSA,
    numDataCols=9, n.compartments=8,
    refLocationProfiles=refLocationProfilesNSA,
    assignPropsMat=protCPAfromNSA,
    yAxisLabel="Normalized Specific Amount")
dev.off()
```

To output plots all of the protein profiles into a single pdf file, one can set up a loop as follows:

```
pdf(file="allPlotsPDFfile.pdf", width=7, height=10)
n.prots <- nrow(protCPAfromNSA)
for (i in 1:n.prots) {
   protPlotfun(protName=rownames(protCPAfromNSA)[i],
       profile=protNSA, numDataCols=9, n.compartments=8,
       refLocationProfiles=refLocationProfilesNSA,
       assignPropsMat=protCPAfromNSA,
       yAxisLabel="Normalized Specific Amount")
}
dev.off()
```

This will result in a single file, `allPlotsPDFfile.pdf`, with a page for each protein plotted. Note that one protein in our data set, `AIF1`, does not have a profile as all peptides were outliers (see Tutorials 5 and 6).

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] protlocassign_0.99.1 lme4_1.1-28          Matrix_1.4-0
#>
#> loaded via a namespace (and not attached):
#>  [1] xfun_0.30           remotes_2.4.2       purrr_0.3.4
#>  [4] splines_4.1.3       lattice_0.20-45     vctrs_0.3.8
#>  [7] testthat_3.1.2      viridisLite_0.4.0   usethis_2.1.5
#> [10] htmltools_0.5.2     yaml_2.3.5          pracma_2.3.8
#> [13] utf8_1.2.2          rlang_1.0.2         pkgbuild_1.3.1
#> [16] pillar_1.7.0        nloptr_2.0.0        glue_1.6.2
#> [19] withr_2.5.0         BiocParallel_1.28.3 sessioninfo_1.2.2
#> [22] lifecycle_1.0.1     outliers_0.14       stringr_1.4.0
#> [25] gtable_0.3.0        devtools_2.4.3      evaluate_0.15
#> [28] memoise_2.0.1       knitr_1.37          callr_3.7.0
#> [31] fastmap_1.1.0       ps_1.6.0            parallel_4.1.3
#> [34] fansi_1.0.2         highr_0.9           Rcpp_1.0.8
#> [37] plot.matrix_1.6.1   cachem_1.0.6        desc_1.4.1
#> [40] pkgload_1.2.4       fs_1.5.2            brio_1.1.3
#> [43] gridExtra_2.3       BB_2019.10-1        digest_0.6.29
#> [46] stringi_1.7.6       processx_3.5.2      rprojroot_2.0.2
#> [49] grid_4.1.3          quadprog_1.5-8      cli_3.2.0
#> [52] tools_4.1.3         magrittr_2.0.2      tibble_3.1.6
```

```
#> [55] crayon_1.5.0        pkgconfig_2.0.3     ellipsis_0.3.2
#> [58] MASS_7.3-55         prettyunits_1.1.1   minqa_1.2.4
#> [61] rmarkdown_2.13      rstudioapi_0.13     R6_2.5.1
#> [64] boot_1.3-28         nlme_3.1-155        compiler_4.1.3
```