Tutorial 2: Relative specific amount (RSA) transformation and CPA

DFM & PL

2022-03-12

Contents

Introduction

In Tutorial 1 ("Getting Started"), we illustrated the principles of constrained proportional assignment using protein profiles that represent mass spectrometry data from a set of subcellular fractions in the form of normalized specific amounts (NSAs). NSA profiles have equivalent amounts of total protein analyzed per fraction, with the sum of all fractions constrained to 1. This tutorial describes how to use functions in the `protlocassign` package to transform NSA profiles in a manner that is more appropriate for inferring proportional residence in subcellular compartments.

Setting up the data and reference protein files, and transforming to Relative Specific Amounts (RSAs)

As explained in the Tutorial 1, we will demonstrate using two R data sets that are included in the `protlocassign` package. One, `protNSA_AT5tmtMS2`, consists of row names that indicate protein identifiers, each of which is followed by data describing the identifier profile across the nine normalized specific amounts derived from a subcellular fractionation experiment. The other data set, `markerListJadot`, consists of a list of reference proteins and their associated known subcellular compartments. As before, to run the program, the `protlocassign` library must be installed.

```
library(protlocassign)
```

In Tutorial 1, we use `protNSA_AT5tmtMS2` and an untransformed average of reference protein profiles in the form of NSAs for each compartment to conduct CPA. However, it may be advantageous to transform the data prior to conducting CPA to yield a more accurate prediction of cellular location. For this purpose, we express profile data as relative specific amounts (RSAs). As explained in the main text and elaborated in Tutorial 3, RSA is the ratio of two ratios: the numerator is the amount of a given protein in a particular fraction divided by the amount of that given protein in the starting material while the denominator is amount of total

protein in a particular fraction divided by the amount of total protein in the starting material. The RSA describes the fold-enrichment (RSA>1) or depletion (RSA<1) of a protein during the fractionation process, and is analogous to the relative specific activity term used in classical analytical subcellular fractionation. Be aware that to perform this transformation, one needs to have estimates of all these quantities, and this was incorporated into our experimental design. In our example, the first six fractions (the differential fractions) can be used to estimate amounts in the starting material. We also measured total protein in each fraction, and these are contained in the 9-element vector `totProtAT5` which is preloaded in `protlocassign`. Note that the order and numbers of the measurements for total protein (e.g., N, M, L1, L2, P, S, Nyc1, Nyc2 and Nyc3 in `totProtAT5`) must correspond to those in the data set containing individual protein profiles (e.g., `protNSA_AT5tmtMS2` or `protNSA_test`). For clarify of presentation, we rename `totProtAT5` and `protProfileNSA_test` to `totProt` and `protProfileNSA`, respectively.

```
data(protNSA_test)
data(totProtAT5)
protNSA <- protNSA_test
str(protNSA)
```

```
#> 'data.frame':    40 obs. of  11 variables:
#>  $ 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 ...
```

```
totProt <- totProtAT5
round(totProt, digits=4)
```

```
#>       N       M      L1      L2       P       S   Nyc.1   Nyc.2   Nyc.3
#> 46.0448 48.9560  1.3841  1.5663 24.0456 58.1818  0.0369  0.0685  1.2730
```

The function `RSAfromNSA` calculates transformed profiles from individual and total protein measurements. This requires specifying which values are used to estimate the amount in the starting material (typically the homogenate) and the values used to construct the profile. In our case, the first six fractions of the nine-fraction profile are summed to estimate the starting material. Our code requires that the fractions representing the starting material are contiguous and are located at the beginning of the profile. Note that the function `RSAfromNSA` can use protein profiles expressed either as NSAs or as specific amounts. Thus we select the first nine columns of `protNSA`:

```
protRSA  <- RSAfromNSA(NSA=protNSA[,1:9],
                       NstartMaterialFractions=6, totProt=totProt)
dim(protRSA)
```

```
#> [1] 40  9
```

```
str(protRSA)
```

```
#> 'data.frame':    40 obs. of  9 variables:
#>  $ N   : num  0.963 NA 1.578 1.024 1.146 ...
#>  $ M   : num  1.036 NA 0.864 0.616 0.688 ...
#>  $ L1  : num  2.075 NA 1.235 0.913 0.972 ...
#>  $ L2  : num  1.77 NA 1.88 2.15 2.49 ...
#>  $ P   : num  0.935 NA 1.609 4.084 3.143 ...
#>  $ S   : num  9.80e-01 NA 3.76e-01 1.31e-16 2.21e-01 ...
#>  $ Nyc1: num  2.632 NA 2.674 4.346 0.867 ...
#>  $ Nyc2: num  8.207 NA 2.477 5.512 0.938 ...
#>  $ Nyc3: num  1.334 NA 0.805 0.268 0.698 ...
```

Since there is additional information in the last two columns of `protNSA` that we want to include in the new file, specifically the numbers of spectra and peptides (`Nspectra` and `Nseq`), we add them to the output as follows:

```
protRSA <- data.frame(protRSA, protNSA[,10:11])
#note data frame is being overwritten
dim(protRSA)
```

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

```
str(protRSA)
```

```
#> 'data.frame':    40 obs. of  11 variables:
#>  $ N       : num  0.963 NA 1.578 1.024 1.146 ...
#>  $ M       : num  1.036 NA 0.864 0.616 0.688 ...
#>  $ L1      : num  2.075 NA 1.235 0.913 0.972 ...
#>  $ L2      : num  1.77 NA 1.88 2.15 2.49 ...
#>  $ P       : num  0.935 NA 1.609 4.084 3.143 ...
#>  $ S       : num  9.80e-01 NA 3.76e-01 1.31e-16 2.21e-01 ...
#>  $ Nyc1    : num  2.632 NA 2.674 4.346 0.867 ...
#>  $ Nyc2    : num  8.207 NA 2.477 5.512 0.938 ...
#>  $ Nyc3    : num  1.334 NA 0.805 0.268 0.698 ...
#>  $ Nspectra: num  41 0 237 2 78 661 17 1690 98 55 ...
#>  $ Npep    : num  8 0 41 2 14 29 6 62 11 22 ...
```

We also need to transform the profiles of the markers for each compartment. As in Tutorial 1, we use the function `locationProfilesetup` to average the profiles (which must be normalized specific amounts) to obtain profiles for the reference proteins:

```
data(markerListJadot)
refLocationProfilesNSA <- locationProfileSetup(profile=protNSA,
                          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
```

We then use `RSAfromNSA` to transform these reference profiles.

```
refLocationProfilesRSA <- RSAfromNSA(NSA=refLocationProfilesNSA, NstartMaterialFractions=6,
       totProt=totProt)
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 computed the RSA reference profiles above from the NSA reference profiles. Note that, as an alternative, one could compute the RSA reference profiles directly from `protRSA`. These two approaches yield similar but non-identical results, and we typically use the first procedure to generate RSA-transformed reference location profiles.

```
refLocationProfilesRSA_2 <- locationProfileSetup(profile=protRSA,
                            markerList=markerListJadot, numDataCols=9)
round(refLocationProfilesRSA_2, digits=4)
```

```
#>            N      M     L1     L2      P      S   Nyc1   Nyc2   Nyc3
#> Cyto  0.4386 0.3667 0.4192 0.4138 0.2928 2.2990 0.7212 0.5972 0.3000
#> ER    1.0527 0.7653 1.0975 2.4211 2.7553 0.3898 0.9365 0.9683 0.8634
#> Golgi 0.9633 0.6396 0.6798 1.1532 3.5198 0.2944 2.1530 2.1981 0.3641
#> Lyso  0.9161 1.0991 2.2919 1.6165 0.8987 0.9776 3.1585 9.7587 1.4113
#> Mito  0.9680 2.1420 1.1846 0.5677 0.3329 0.3473 0.5965 0.5206 0.9100
#> Nuc   3.4162 0.1458 0.0717 0.1614 0.1748 0.1923 0.1733 0.1799 0.1299
#> Perox 0.9190 1.2101 4.9567 2.4711 0.6772 0.8870 1.1719 1.4840 3.8301
#> PM    1.5167 0.8482 1.0945 1.5684 1.4883 0.4995 3.2280 2.3926 0.6922
```

```
# we use the `as.matrix` function for display purposes in the tutorial
as.matrix(all.equal(refLocationProfilesRSA, refLocationProfilesRSA_2,
                    precision=0, countEQ=TRUE))
```

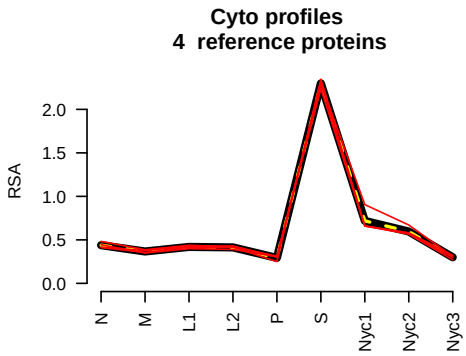
```
#>       [,1]
#>  [1,] "Component \"N\": Mean relative difference: 0.004527338"
#>  [2,] "Component \"M\": Mean relative difference: 0.004945014"
#>  [3,] "Component \"L1\": Mean relative difference: 0.01390615"
#>  [4,] "Component \"L2\": Mean relative difference: 0.02442931"
```

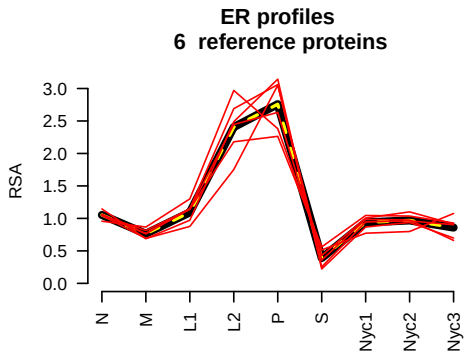
```
#>  [5,] "Component \"P\": Mean relative difference: 0.008787258"
#>  [6,] "Component \"S\": Mean relative difference: 0.01134001"
#>  [7,] "Component \"Nyc1\": Mean relative difference: 0.03760237"
#>  [8,] "Component \"Nyc2\": Mean relative difference: 0.0385269"
#>  [9,] "Component \"Nyc3\": Mean relative difference: 0.01239751"
```

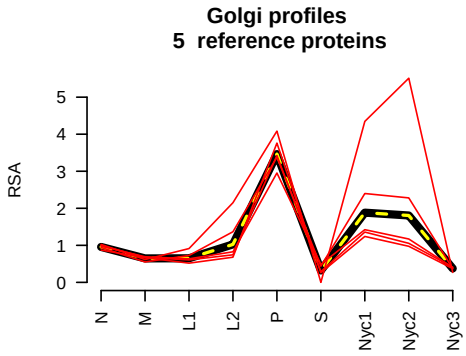
Plotting RSA-transformed profiles, and finding RSA-based constrained proportional assignments

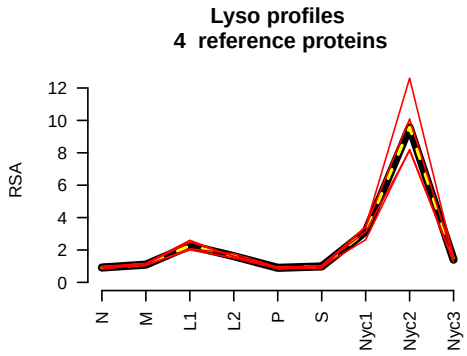
As in Tutorial 1, we can plot reference profiles, but this time using the RSA-transformed data. For example, here is a plot for all markers:

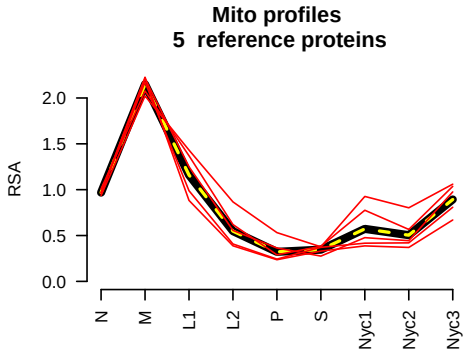
```
loc.list <- rownames(refLocationProfilesRSA)
n.loc <- length(loc.list)
par(mfrow=c(4,2))
for (i in 1:n.loc) {
  markerProfilePlot(refLoc=loc.list[i], profile=protRSA,
                    markerList=markerListJadot,
                    refLocationProfiles=refLocationProfilesRSA, ylab="RSA")
  }
```

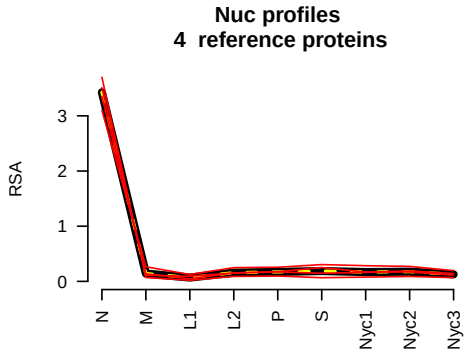

Cyto profiles
4 reference proteins
RSA
2.0
1.5
1.0
0.5
0.0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3

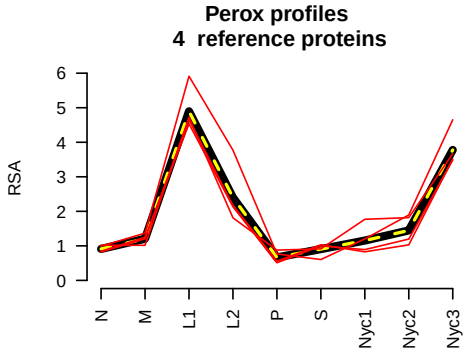

ER profiles
6 reference proteins
RSA
3.0
2.5
2.0
1.5
1.0
0.5
0.0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3

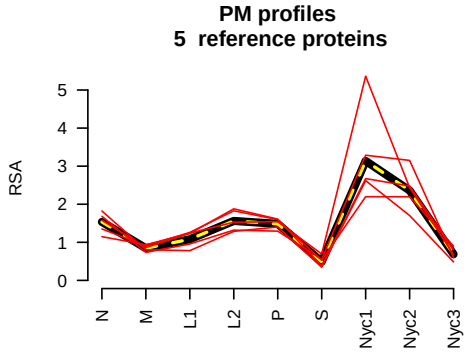

Golgi profiles
5 reference proteins
RSA
5
4
3
2
1
0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Lyso profiles
4 reference proteins
RSA
12
10
8
6
4
2
0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Mito profiles
5 reference proteins
RSA
2.0
1.5
1.0
0.5
0.0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Nuc profiles
4 reference proteins
RSA
3
2
1
0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


Perox profiles
4 reference proteins
RSA
6
5
4
3
2
1
0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3


PM profiles
5 reference proteins
RSA
5
4
3
2
1
0
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3

Now we can run the CPA routine on the RSA-transformed levels; note that this may take several minutes to complete. The result is a matrix with protein identifiers as row names, and data indicating the estimated proportional assignments of each protein among the eight subcellular locations.

```
protCPAfromRSA <- fitCPA(profile=protRSA,
                         refLocationProfiles=refLocationProfilesRSA,
                         numDataCols=9)
```

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

```
str(protCPAfromRSA)
```

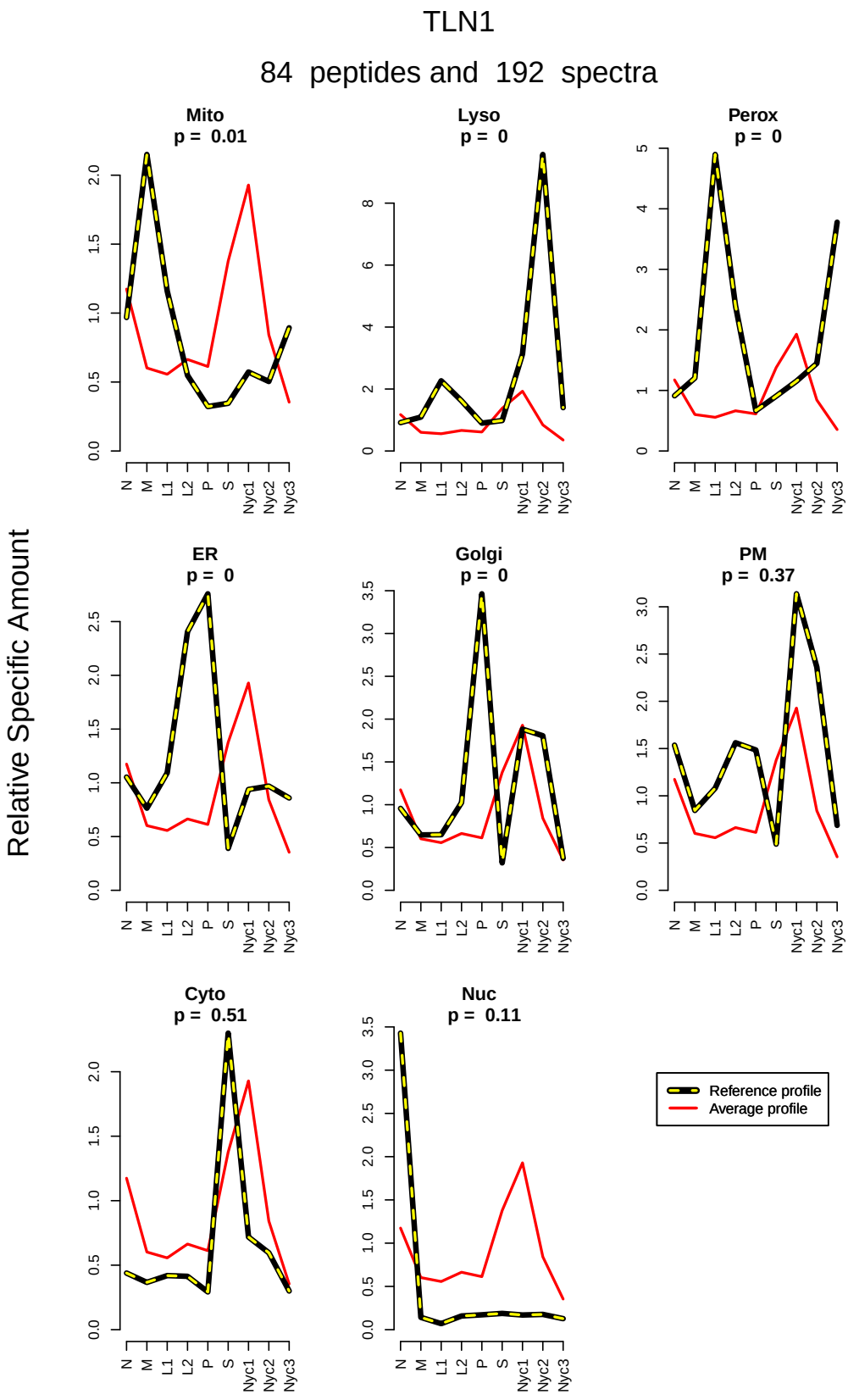
```
#> 'data.frame':    40 obs. of  10 variables:
#>  $ Cyto     : num  0.0341 NA 0 0 0 ...
#>  $ ER       : num  0.0842 NA 0.1738 0 0.9537 ...
#>  $ Golgi    : num  0 NA 0 0.5756 0.0463 ...
#>  $ Lyso     : num  0.8399 NA 0.0553 0.4244 0 ...
#>  $ Mito     : num  0 NA 0 0 0 ...
#>  $ Nuc      : num  0.0207 NA 0.0165 0 0 ...
#>  $ Perox    : num  0.0211 NA 0.0232 0 0 ...
#>  $ PM       : num  0 NA 0.731 0 0 ...
#>  $ Nspectra : num  41 0 237 2 78 661 17 1690 98 55 ...
#>  $ Npeptides: num  8 0 41 2 14 29 6 62 11 22 ...
```

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

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.

Now we plot the results for protein TLN1:

```
protPlotfun(protName="TLN1", profile=protRSA, numDataCols=9,
                         refLocationProfiles=refLocationProfilesRSA,
                         assignPropsMat=protCPAfromRSA,
                         yAxisLabel="Relative Specific Amount")
```


TLN1
84 peptides and 192 spectra
Relative Specific Amount
Mito
p = 0.01
Lyso
p = 0
Perox
p = 0
ER
p = 0
Golgi
p = 0
PM
p = 0.37
Cyto
p = 0.51
Nuc
p = 0.11
N
M
L1
L2
P
S
Nyc1
Nyc2
Nyc3
Reference profile
Average profile

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 35 percent residence proportion to plasma membrane and 53 percent residence to cytosol. As in Tutorial 1, the observed red profile is a weighted mixture of the expected yellow-black lines.

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] Rcpp_1.0.8          lattice_0.20-45     prettyunits_1.1.1
#>  [4] ps_1.6.0            rprojroot_2.0.2     digest_0.6.29
#>  [7] utf8_1.2.2          R6_2.5.1            evaluate_0.15
#> [10] pracma_2.3.8        highr_0.9           pillar_1.7.0
#> [13] rlang_1.0.2         rstudioapi_0.13     minqa_1.2.4
#> [16] callr_3.7.0         nloptr_2.0.0        rmarkdown_2.13
#> [19] desc_1.4.1          devtools_2.4.3      splines_4.1.3
#> [22] BiocParallel_1.28.3 stringr_1.4.0       plot.matrix_1.6.1
#> [25] tinytex_0.37        compiler_4.1.3      xfun_0.30
#> [28] pkgconfig_2.0.3     pkgbuild_1.3.1      htmltools_0.5.2
#> [31] tibble_3.1.6        gridExtra_2.3       BB_2019.10-1
```

```
#> [34] 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
#> [40] brio_1.1.3          grid_4.1.3          nlme_3.1-155
#> [43] gtable_0.3.0        lifecycle_1.0.1     magrittr_2.0.2
#> [46] cli_3.2.0           stringi_1.7.6       cachem_1.0.6
#> [49] fs_1.5.2            remotes_2.4.2       testthat_3.1.2
#> [52] ellipsis_0.3.2      vctrs_0.3.8         boot_1.3-28
#> [55] tools_4.1.3         outliers_0.14       glue_1.6.2
#> [58] purrr_0.3.4         processx_3.5.2      pkgload_1.2.4
#> [61] parallel_4.1.3      fastmap_1.1.0       yaml_2.3.5
#> [64] sessioninfo_1.2.2   memoise_2.0.1       knitr_1.37
#> [67] usethis_2.1.5
```