Tutorial 3: Data transformations: Notation and mixtures using protlocassign

DFM & PL

2022-03-12

Contents

Introduction

As explained in the main text and in Tutorial 2, there are different ways to transform protein profile data. Here, we describe a way to explore the effect of using transformed data with CPA. Briefly, we conduct different data transformations on a set of theoretical proteins that have a range of distributions between two cellular compartments, and then conduct CPA and determine how well it predicts the original distribution. To create the theoretical proteins for our simulations, we use data from the experiment from Tannous et al which consists of a TMT-MS2 analysis of six differential fraction (N, M, L1, L2, P, and S) obtained from centrifugation of a rat liver homogenate and three fractions from a Nycodenz density gradient separation of the differential fraction L1 (Nyc1, Nyc2, and Nyc3).

In our procedure, we first use the eight compartment profiles generated from the marker protein set to simulate a set of eight theoretical proteins that wholly reside in each of the respective compartments. We then then create binary mixtures with defined combinations of the eight theoretical proteins to simulate proteins that are distributed in varying proportions between two compartments. Note that data from mass spectrometry experiments generally represent specific amounts $s_{\alpha,l}$ of a protein α in fraction l, with the same amount of total protein being analyzed for each sample (fraction). For conducting our simulations, we first must transform this data into relative amounts, so that each protein has precisely the same total amount in the initial starting material used for fractionation.

Computation of Relative Amounts

Consider an n by 9 matrix of specific amounts that details the average distribution of n proteins among 9 fractions:

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{19} \\ s_{21} & s_{22} & \cdots & s_{29} \\ \vdots & \vdots & & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{n9} \end{bmatrix}$$

Each row α represents a mean profile for a protein α, with each profile consisting of $f = 9$ fractions.

In the Tutorial 1 we actually used a normalized specific amount (NSA), denoted here as $\tilde{s}_{\alpha,l}$ calculated as follows:

$$\tilde{s}_{\alpha,l} = \frac{s_{\alpha,l}}{\sum_{j=1}^{f} s_{\alpha,j}}$$

In some experiments, these are the only values available for using the CPA procedure. However, with appropriate experimental design and execution, using balance sheet analysis ("bookkeeping"), one can estimate the amounts of total protein present in different samples obtained from a set amount of starting material. These include the total protein content of the starting material (designated t_h) and the total protein content of any given fraction (designated t_l for fraction l). We can use these to calculate the amount of protein in arbitrary units in fraction l derived from a set amount of starting material as $a_l = s_l t_l$. Note that as a_l is in arbitrary units one can do the same calculation using either s_l or $\tilde{s}_l$, as later, these will yield the same values when calculating relative amounts (see below).

To see how this works in `protlocassign`, let us consider the Jadot et al. reference protein profiles which we generate using the `locationProfileSetup` function. As in earlier tutorials, we rename `protNSA_test` as `protNSA` and we tailor the output using the `round` function.

```
library(protlocassign)
data(protNSA_test)
data(markerListJadot)
protNSA <- protNSA_test
refLocationProfilesNSA <- locationProfileSetup(profile=protNSA,
                          markerList=markerListJadot, numDataCols=9)
round(refLocationProfilesNSA, digits=3)
```

```
#>           N     M    L1    L2     P     S  Nyc1  Nyc2  Nyc3
#> Cyto  0.075 0.063 0.072 0.071 0.050 0.393 0.123 0.102 0.051
#> ER    0.094 0.068 0.097 0.214 0.246 0.035 0.084 0.086 0.076
#> Golgi 0.086 0.058 0.058 0.092 0.311 0.029 0.169 0.162 0.034
#> Lyso  0.042 0.050 0.104 0.074 0.041 0.045 0.143 0.437 0.064
#> Mito  0.130 0.288 0.155 0.073 0.043 0.046 0.077 0.068 0.120
#> Nuc   0.741 0.031 0.015 0.034 0.037 0.041 0.037 0.038 0.027
#> Perox 0.052 0.069 0.282 0.138 0.038 0.052 0.066 0.083 0.218
#> PM    0.117 0.064 0.082 0.118 0.113 0.037 0.238 0.179 0.052
```

Here, each row represents the profile $\tilde{s}_l$ for a protein resident solely in a particular cellular compartment. The amount of total protein derived from a set amount of starting material from all fractions in the experiment used for these tutorials is in `totProtAT5`, a vector supplied with the `protassign` package. For convenience we rename it `totProt`.

```
data(totProtAT5)
totProt <- totProtAT5
round(totProt, digits=3)
```

```
#>       N      M     L1     L2      P      S  Nyc.1  Nyc.2  Nyc.3
#> 46.045 48.956  1.384  1.566 24.046 58.182  0.037  0.068  1.273
```

We denote these values using a vector $\mathbf{t} = (t_1, t_2, \ldots, t_9)$.

As noted above, for protein α in fraction l, we have $a_{\alpha,l} = \tilde{s}_{\alpha,l} t_l$. In matrix form,

$$\mathbf{A} = \widetilde{\mathbf{S}} \cdot \mathrm{diag}(\mathbf{t}) = \begin{bmatrix} \widetilde{s}_{11}t_1 & \widetilde{s}_{12}t_2 & \cdots & \widetilde{s}_{19}t_9 \\ \widetilde{s}_{21}t_1 & \widetilde{s}_{22}t_2 & \cdots & \widetilde{s}_{29}t_9 \\ \vdots & \vdots & & \vdots \\ \widetilde{s}_{n1}t_1 & \widetilde{s}_{n2}t_2 & \cdots & \widetilde{s}_{n9}t_9 \end{bmatrix}$$

We need this matrix to do the next step, which is to convert to a common scale for all proteins. We do this by normalizing to the amount of protein α in the starting material, which we denote as $a_{\alpha,h}$. If the homogenate is measured directly, this can be calculated from $\widetilde{s}_{\alpha,h} t_h$. Alternatively, if a complete set of fractions that entirely represent the homogenate are available, it is preferable to calculate this by summing $\widetilde{s}_{\alpha,l} t_l$ over these fractions. In our case, the first six fractions (N, M, L1, L2, P, and S) are a complete set of differential fractions that represent the starting material, and thus

$$a_{\alpha,h} = \sum_{i=1}^{6} \tilde{s}_{\alpha,i} t_i$$

and $t_h = \sum_{i=1}^{6} t_i$. Note that this is readily calculated by summing the first six elements of $\mathbf{t}$.

```
sum(totProt[1:6])
```

```
#> [1] 180.1785
```

(The last three "Nyc" columns were derived from L1, so we do not include them in the sum.)

Finally, we normalize amounts in any given fraction to amounts in starting material, which we call the relative amount, designated here as `Acup`, and denote as $\breve{a}_{\alpha,l}$

In our example,

$$\breve{a}_{\alpha,l} = \frac{a_{\alpha,l}}{a_{\alpha,h}} = \frac{s_{\alpha,l} t_l}{s_{\alpha,h} t_h} = \frac{s_{\alpha,l} t_l}{\sum_{i=1}^{6} s_{\alpha,i} t_i} = \frac{\tilde{s}_{\alpha,l} t_l}{\sum_{i=1}^{6} \tilde{s}_{\alpha,i} t_i}$$

In matrix form, we may write this as:

$$\breve{\mathbf{A}} = \mathbf{A} \cdot \mathrm{diag}(1/a_{\alpha,h}) = \begin{bmatrix} \tilde{s}_{11}t_1/a_{1,h} & \tilde{s}_{12}t_2/a_{1,h} & \cdots & \tilde{s}_{19}t_9/a_{1,h} \\ \tilde{s}_{21}t_1/a_{2,h} & \tilde{s}_{22}t_2/a_{2,h} & \cdots & \tilde{s}_{29}t_9/a_{2,h} \\ \vdots & \vdots & & \vdots \\ \tilde{s}_{n1}t_1/a_{n,h} & \tilde{s}_{n2}t_2/a_{n,h} & \cdots & \tilde{s}_{n9}t_9/a_{n,h} \end{bmatrix}$$

or simply as $\breve{\mathbf{A}} = [\breve{a}_{\alpha,l}]$.

The function `AcupFromNSA` computes this:

```
refLocationProfilesAcup <- AcupFromNSA(NSA=refLocationProfilesNSA, NstartMaterialFractions=6,
                                       totProt=totProt)
round(refLocationProfilesAcup, digits=4)
```

```
#>            N      M     L1     L2      P      S  Nyc1   Nyc2   Nyc3
#> Cyto  0.1121 0.0996 0.0032 0.0036 0.0390 0.7424 1e-04 0.0002 0.0021
#> ER    0.2691 0.2075 0.0084 0.0209 0.3681 0.1259 2e-04 0.0004 0.0061
#> Golgi 0.2445 0.1758 0.0050 0.0089 0.4623 0.1035 4e-04 0.0007 0.0026
#> Lyso  0.2341 0.2972 0.0174 0.0141 0.1199 0.3173 6e-04 0.0036 0.0099
#> Mito  0.2474 0.5842 0.0089 0.0048 0.0430 0.1117 1e-04 0.0002 0.0063
#> Nuc   0.8761 0.0385 0.0005 0.0014 0.0229 0.0606 0e+00 0.0001 0.0009
#> Perox 0.2329 0.3273 0.0376 0.0209 0.0888 0.2925 2e-04 0.0005 0.0267
#> PM    0.3925 0.2296 0.0083 0.0136 0.1982 0.1578 6e-04 0.0009 0.0049
```

The values in `refLocationProfilesAcup` represent in principle the relative amount of a given cellular compartment that ends up in a given centrifugation fraction.

Computation of Relative Specific Amounts from Relative Amounts

When examining the distribution of a given protein in different fractions, it is particularly useful to consider its abundance relative to that of total protein. We refer to this as a Relative Specific Amount (RSA or r), which can be calculated as $r_{\alpha,l} = \breve{a}_{\alpha,l}/\breve{t}_l$, where the vector $\breve{\mathbf{t}} = \frac{\mathbf{t}}{t_h} = \frac{\mathbf{t}}{\sum_1^6 t_i}$. For a protein α the RSA in fraction l is given by:

$$r_{\alpha,l} = \frac{\breve{a}_{\alpha,l}}{\breve{t}_l} = \frac{\frac{\widetilde{s}_{\alpha,l} t_l}{\sum_{i=1}^6 \widetilde{s}_{\alpha,i} t_i}}{\frac{t_l}{\sum_{i=1}^6 t_i}} = \frac{s_{\alpha,l} \cdot \sum_{i=1}^6 t_i}{\sum_{i=1}^6 s_{\alpha,i} t_i} = \frac{\widetilde{s}_{\alpha,l} \cdot \sum_{i=1}^6 t_i}{\sum_{i=1}^6 \widetilde{s}_{\alpha,i} t_i}$$

In matrix form, this is $\mathbf{R} = [r_{\alpha,l}]$.

We can get the RSA matrix from `refLocationProfilesAcup` and `totProt` as follows:

```
refLocationProfilesRSA <- RSAfromAcup(refLocationProfilesAcup, NstartMaterialFractions=6, totProt=totPr
round(refLocationProfilesRSA, digits=3)
```

```
#>           N     M    L1    L2     P     S  Nyc1  Nyc2  Nyc3
#> Cyto  0.439 0.366 0.419 0.413 0.293 2.299 0.718 0.596 0.300
#> ER    1.053 0.764 1.093 2.406 2.758 0.390 0.938 0.969 0.859
#> Golgi 0.957 0.647 0.650 1.024 3.464 0.320 1.882 1.806 0.375
#> Lyso  0.916 1.094 2.266 1.619 0.899 0.983 3.131 9.571 1.399
#> Mito  0.968 2.150 1.160 0.548 0.322 0.346 0.573 0.504 0.893
#> Nuc   3.428 0.142 0.069 0.158 0.171 0.188 0.169 0.176 0.127
#> Perox 0.911 1.205 4.897 2.400 0.666 0.906 1.153 1.446 3.778
#> PM    1.536 0.845 1.081 1.562 1.485 0.489 3.137 2.362 0.687
```

Note that we can also obtain the RSA transformed data directly from NSA data as described in Tutorial 2:

```
refLocationProfilesRSA_2 <- RSAfromNSA(NSA=refLocationProfilesNSA,
                                       NstartMaterialFractions=6, totProt=totProt)
```

This yields precisely the same values as calculated previously using `locationProfileSetup`.

```
identical(refLocationProfilesRSA, refLocationProfilesRSA_2)
```

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

Finally, if we normalize an RSA profile so that the rows sum to one, this yields a normalized specific amount profile (see Appendix of main paper). Consider example the matrix `refLocationProfilesRSA`, which contains the RSA-transformed compartment profiles. We normalize the rows using the `apply` function, and then transpose using the `t` function to yield a matrix of the normalized specific amounts; these values are essentially identical to those that we started with in `refLocationProfiles`. This is performed using the function `NSAfromRSA`:

```
refLocationProfilesNSA_2 <- NSAfromRSA(refLocationProfilesRSA_2)
```

Note that `refLocationProfilesNSA_2` is not identical to the values obtained previously using the `locationProfileSetup` function with the protein profiles containing NSA data because of internal precision issues, but both are essentially equivalent.

```
as.matrix(all.equal(refLocationProfilesNSA_2, refLocationProfilesNSA,
                    tolerance=0, countEQ=TRUE))
```

```
#>       [,1]
#>  [1,] "Component \"N\": Mean relative difference: 1.163502e-16"
#>  [2,] "Component \"M\": Mean relative difference: 7.028508e-17"
#>  [3,] "Component \"L1\": Mean relative difference: 5.583468e-17"
#>  [4,] "Component \"L2\": Mean relative difference: 7.659441e-17"
#>  [5,] "Component \"P\": Mean relative difference: 3.535861e-17"
#>  [6,] "Component \"S\": Mean relative difference: 5.116976e-17"
#>  [7,] "Component \"Nyc1\": Mean relative difference: 9.185514e-17"
#>  [8,] "Component \"Nyc2\": Mean relative difference: 5.309233e-17"
#>  [9,] "Component \"Nyc3\": Mean relative difference: 7.562426e-17"
```

The available transformation functions are `AcupFromNSA`, `RSAfromAcup`, `RSAfromNSA` (a combination of the previous two), and `NSAfromRSA`. For completeness, we also include the functions `NSAfromAcup` and `AcupFromRSA`. All functions except `NSAfromRSA` require arguments for `NstartMaterialFractions` and `totProt`. These functions allow any profile to be expressed in the form of `NSA`, `Acup`, and `RSA`.

Simulating proteins resident in multiple subcellular locations

We may simulate data from proteins with multiple residences using the `proteinMix` function. For example, to simulate data from proteins that are distributed in a range of proportions between cytosol and lysosomes, we use this function with relative amounts of their single-compartment profiles. Note that we need to use Acup-transformed data to create the mixtures so that the total amount of the given protein summed across all fractions will be invariant for all mixtures. By default, we vary the proportions by increments of 0.1, but different values can be specified using the argument `increment.prop` (e.g., `increment.prop=0.2` or `increment.prop=0.05`). We specify the mixing locations by the arguments `Loc1=1` and `Loc2=4`, which are the row numbers of the desired compartment profiles in `refLocationProfiles`:

```
refLocationProfilesAcup <- AcupFromNSA(NSA=refLocationProfilesNSA,
                                       NstartMaterialFractions=6,
                                       totProt=totProt)
data.frame(rownames(refLocationProfilesAcup))
```

```
#>   rownames.refLocationProfilesAcup.
#> 1                              Cyto
#> 2                                ER
#> 3                             Golgi
#> 4                              Lyso
#> 5                              Mito
#> 6                               Nuc
#> 7                             Perox
#> 8                                PM
```

```
mixCytoLysoAcup <- proteinMix(AcupRef=refLocationProfilesAcup,
                              increment.prop=0.1,
                              Loc1=1, Loc2=4)
# Note that the default value of increment.prop=0.1.
# This does not need to be explicitly
#  specified unless a different increment is desired.
```

The result is a matrix that contains the Acup values (relative amounts) for the simulated proteins:

```
round(mixCytoLysoAcup, digits=3)
```

```
#>                        N     M    L1    L2     P     S  Nyc1  Nyc2  Nyc3
#> 0_Cyto:1_Lyso      0.234 0.297 0.017 0.014 0.120 0.317 0.001 0.004 0.010
#> 0.1_Cyto:0.9_Lyso  0.222 0.277 0.016 0.013 0.112 0.360 0.001 0.003 0.009
#> 0.2_Cyto:0.8_Lyso  0.210 0.258 0.015 0.012 0.104 0.402 0.001 0.003 0.008
#> 0.3_Cyto:0.7_Lyso  0.198 0.238 0.013 0.011 0.096 0.445 0.000 0.003 0.008
#> 0.4_Cyto:0.6_Lyso  0.185 0.218 0.012 0.010 0.088 0.487 0.000 0.002 0.007
#> 0.5_Cyto:0.5_Lyso  0.173 0.198 0.010 0.009 0.079 0.530 0.000 0.002 0.006
#> 0.6_Cyto:0.4_Lyso  0.161 0.179 0.009 0.008 0.071 0.572 0.000 0.002 0.005
#> 0.7_Cyto:0.3_Lyso  0.149 0.159 0.007 0.007 0.063 0.615 0.000 0.001 0.004
#> 0.8_Cyto:0.2_Lyso  0.137 0.139 0.006 0.006 0.055 0.657 0.000 0.001 0.004
#> 0.9_Cyto:0.1_Lyso  0.124 0.119 0.005 0.005 0.047 0.700 0.000 0.001 0.003
#> 1_Cyto:0_Lyso      0.112 0.100 0.003 0.004 0.039 0.742 0.000 0.000 0.002
```

Then we can test the CPA algorithm by first converting this mixture data to RSAs:

```
mixCytoLysoRSA <- RSAfromAcup(Acup=mixCytoLysoAcup,
                              NstartMaterialFractions=6, totProt=totProt)

round(mixCytoLysoRSA, digits=3)
```

```
#>                        N     M    L1    L2     P     S  Nyc1  Nyc2  Nyc3
#> 0_Cyto:1_Lyso      0.916 1.094 2.266 1.619 0.899 0.983 3.131 9.571 1.399
#> 0.1_Cyto:0.9_Lyso  0.868 1.021 2.081 1.498 0.838 1.114 2.890 8.673 1.289
#> 0.2_Cyto:0.8_Lyso  0.821 0.948 1.896 1.378 0.778 1.246 2.648 7.776 1.179
#> 0.3_Cyto:0.7_Lyso  0.773 0.876 1.712 1.257 0.717 1.378 2.407 6.878 1.069
#> 0.4_Cyto:0.6_Lyso  0.725 0.803 1.527 1.137 0.656 1.509 2.166 5.981 0.959
#> 0.5_Cyto:0.5_Lyso  0.677 0.730 1.342 1.016 0.596 1.641 1.925 5.083 0.849
#> 0.6_Cyto:0.4_Lyso  0.630 0.657 1.158 0.896 0.535 1.773 1.683 4.186 0.739
#> 0.7_Cyto:0.3_Lyso  0.582 0.585 0.973 0.775 0.474 1.904 1.442 3.288 0.630
#> 0.8_Cyto:0.2_Lyso  0.534 0.512 0.788 0.655 0.414 2.036 1.201 2.391 0.520
#> 0.9_Cyto:0.1_Lyso  0.487 0.439 0.604 0.534 0.353 2.168 0.960 1.493 0.410
#> 1_Cyto:0_Lyso      0.439 0.366 0.419 0.413 0.293 2.299 0.718 0.596 0.300
```

Finally, we fit the CPA algorithm to this RSA-transformed, simulated data using the previously generated RSA-transformed marker protein profiles:

```
mixCytoLysoCPAfromRSA <- fitCPA(profile=mixCytoLysoRSA,
                                refLocationProfiles=refLocationProfilesRSA,
                               numDataCols=9)
round(mixCytoLysoCPAfromRSA, digits=3)
```

```
#>                   Cyto ER Golgi Lyso Mito Nuc Perox PM
#> 0_Cyto:1_Lyso      0.0  0     0  1.0    0   0     0  0
#> 0.1_Cyto:0.9_Lyso  0.1  0     0  0.9    0   0     0  0
#> 0.2_Cyto:0.8_Lyso  0.2  0     0  0.8    0   0     0  0
#> 0.3_Cyto:0.7_Lyso  0.3  0     0  0.7    0   0     0  0
#> 0.4_Cyto:0.6_Lyso  0.4  0     0  0.6    0   0     0  0
#> 0.5_Cyto:0.5_Lyso  0.5  0     0  0.5    0   0     0  0
#> 0.6_Cyto:0.4_Lyso  0.6  0     0  0.4    0   0     0  0
#> 0.7_Cyto:0.3_Lyso  0.7  0     0  0.3    0   0     0  0
#> 0.8_Cyto:0.2_Lyso  0.8  0     0  0.2    0   0     0  0
#> 0.9_Cyto:0.1_Lyso  0.9  0     0  0.1    0   0     0  0
#> 1_Cyto:0_Lyso      1.0  0     0  0.0    0   0     0  0
```

The estimated proportions correspond closely to the proportions used in the simulation (see below).

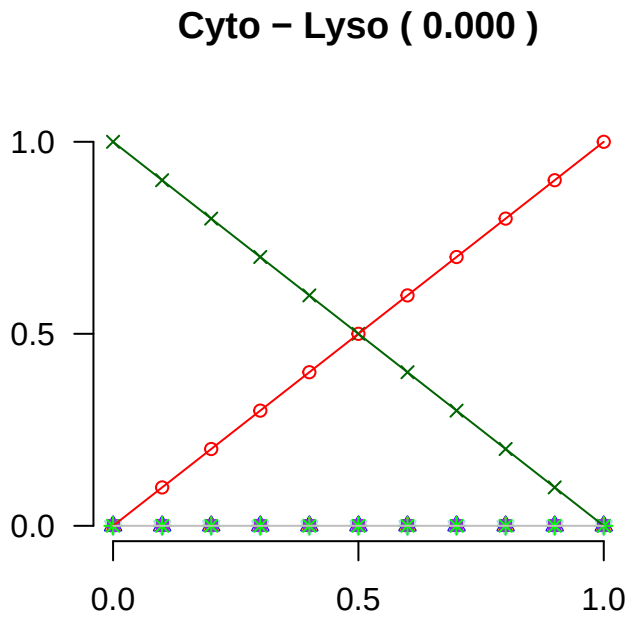
Plotting these estimates against the "true" distributions of the simulated proteins provides insights into different transformations on the goodness of fit. As an introduction, we first illustrate this using plots of simulated proteins distributed in varying amounts between the cytoplasm and the lysosome with CPA conducted on the RSA-transformed data. We then extend this to simulated proteins distributed cytoplasm and each of the other compartments. In Tutorial 4, we use the simulated mixtures to evaluate the effect of various transformations on the CPA procedure.

Plotting mixtures of proteins with transformations

We can use the `mixturePlot` function to evaluate the effect of different data transformations used to represent protein profiles on the compartmental distributions estimated by CPA. The function produces graphical representations by plotting the theoretical distribution (based on simulation parameters, x-coordinate) versus the predicted values (based on CPA, y-coordinate).

This function also evaluates the prediction error by computing the area separating the predicted and expected protein mixtures via the trapezoidal rule; this is done with the `trapz` function in the `pracma` library, which must have been previously installed. We also need to tell the program which two locations were used to generate the mixtures using `Loc1` and `Loc2`. As our first example, we examine the results CPA using the RSA transformation on simulated proteins distributed between Cyto and Lyso.

```
library(pracma)
par(mfrow=c(1,1))  # reset window for a single plot
# The argument increment.prop needs to match the value used
#   in creating the mixture using proteinMix. This does
#   not need to be specified if using the value of 0.1
mixturePlot(mixProtiProtjCPA=mixCytoLysoCPAfromRSA,
            NstartMaterialFractions=6, Loc1=1, Loc2=4,
            increment.prop=0.1, xaxisLab=TRUE, yaxisLab=TRUE)
```

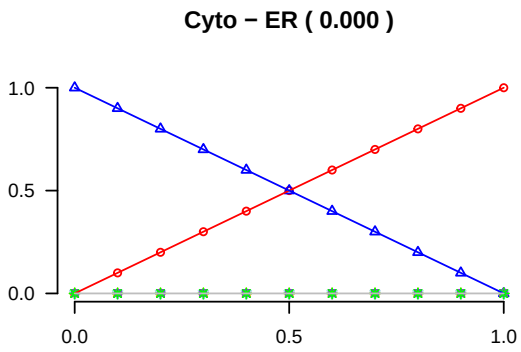


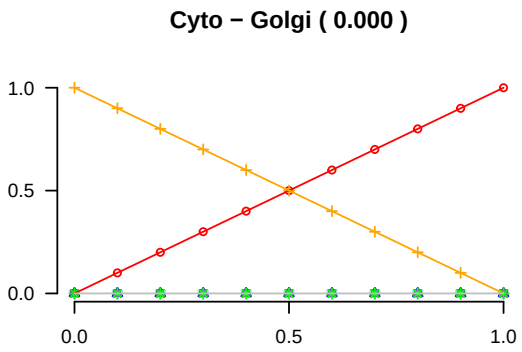
Here we see visually that the estimated proportions match the simulated ones. The prediction error, the area separated by the observed and expected CPA estimates, is nearly zero, and is shown in parentheses.

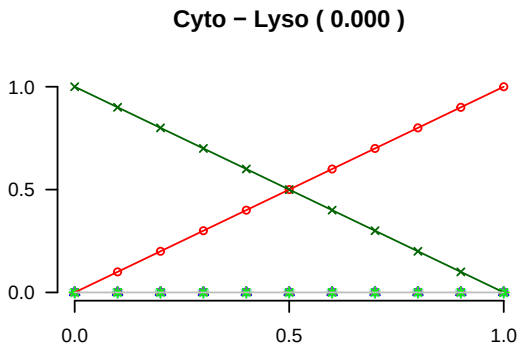
Next, we consider a mixture of Cyto with each of the other seven compartments using RSA-based transformations. We begin by setting up the plot area for a 4 by 2 array of plots. Optionally, to control the size of the window, we may want to explicitly open a window using `windows(height=10, width=7)`. Next we fix one component of the mixture to the first, which is Cyto, and loop over the other 7 subcellular compartments, creating mixtures of the `refLocationProfilesAcup` values. For each case, we transform these mixtures to RSA and obtain CPA mixing proportion estimates from these RSA-transformed mixtures and compute the area-based prediction errors. These values are stored in a data frame `mixErrorMat` which is then renamed to avoid overwriting since multiple mixtures and transformations are being explored.

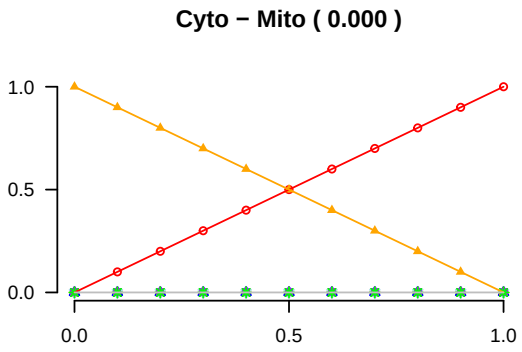
```
par(mfrow=c(4,2))
i <- 1
mixErrorMat <- NULL
for (j in 2:8) {
   # Create the mixture of Cyto (i = 1) with compartment j
   mixProtiProtjAcup <- proteinMix(Acup=refLocationProfilesAcup,
                                   Loc1=i, Loc2=j)

   # Tranform the mixtures to relative specific amounts
   mixProtiProtjRSA <- RSAfromAcup(Acup=mixProtiProtjAcup,
                        NstartMaterialFractions=6, totProt=totProt)

   # Find the constrained proportional assigments (CPA)
   mixProtiProtjCPAfromRSA <- fitCPA(profile=mixProtiProtjRSA,
                    refLocationProfiles=refLocationProfilesRSA,
                    numDataCols=9)

   # Plot the results, including the area-based error estimate,
```

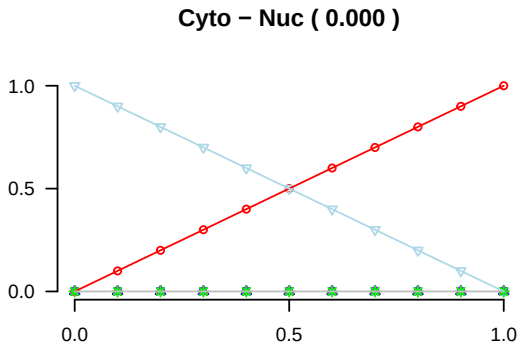
```
   #    and collect the area-based errors (errorReturn=TRUE)
    mixResult <- mixturePlot(mixProtiProtjCPA=mixProtiProtjCPAfromRSA,
                             NstartMaterialFractions=6, Loc1=i, Loc2=j,
                             increment.prop=0.1, errorReturn = TRUE)
    mixErrorMat <- rbind(mixErrorMat, mixResult)
}
mixErrorAllCytoRSA <- mixErrorMat
```

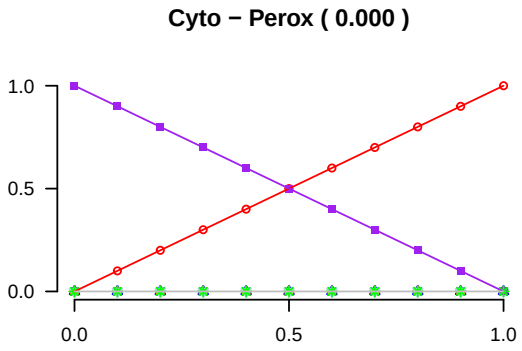

Cyto − ER (0.000)
1.0
0.5
0.0
0.0
0.5
1.0

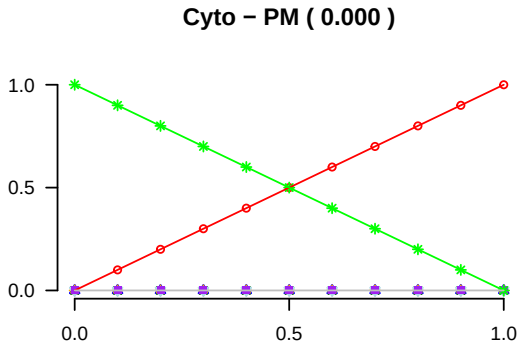

Cyto − Golgi (0.000)
1.0
0.5
0.0
0.0
0.5
1.0


Cyto − Lyso (0.000)
1.0
0.5
0.0
0.0
0.5
1.0


Cyto − Mito (0.000)
1.0
0.5
0.0
0.0
0.5
1.0


Cyto − Nuc (0.000)
1.0
0.5
0.0
0.0
0.5
1.0


Cyto − Perox (0.000)
1.0
0.5
0.0
0.0
0.5
1.0


Cyto − PM (0.000)
1.0
0.5
0.0
0.0
0.5
1.0

All seven mixtures have essentially zero area-based error, as we expect. We can examine these errors with more precision as follows:

```
mixErrorAllCytoRSA
```

```
#>   Loc1  Loc2    ErrorArea
#> 1 Cyto    ER 2.974605e-06
#> 2 Cyto Golgi 2.071270e-06
#> 3 Cyto  Lyso 9.154619e-07
#> 4 Cyto  Mito 3.297390e-07
#> 5 Cyto   Nuc 2.196890e-07
#> 6 Cyto Perox 7.449870e-07
#> 7 Cyto    PM 3.047688e-06
```

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] pracma_2.3.8         protlocassign_0.99.1 lme4_1.1-28
#> [4] 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] highr_0.9           pillar_1.7.0        rlang_1.0.2
#> [13] rstudioapi_0.13     minqa_1.2.4         callr_3.7.0
#> [16] nloptr_2.0.0        rmarkdown_2.13      desc_1.4.1
#> [19] devtools_2.4.3      splines_4.1.3       BiocParallel_1.28.3
#> [22] stringr_1.4.0       plot.matrix_1.6.1   tinytex_0.37
#> [25] compiler_4.1.3      xfun_0.30           pkgconfig_2.0.3
#> [28] pkgbuild_1.3.1      htmltools_0.5.2     tibble_3.1.6
#> [31] gridExtra_2.3       BB_2019.10-1        quadprog_1.5-8
#> [34] fansi_1.0.2         viridisLite_0.4.0   crayon_1.5.0
#> [37] withr_2.5.0         MASS_7.3-55         brio_1.1.3
#> [40] grid_4.1.3          nlme_3.1-155        gtable_0.3.0
#> [43] lifecycle_1.0.1     magrittr_2.0.2      cli_3.2.0
#> [46] stringi_1.7.6       cachem_1.0.6        fs_1.5.2
#> [49] remotes_2.4.2       testthat_3.1.2      ellipsis_0.3.2
#> [52] vctrs_0.3.8         boot_1.3-28         tools_4.1.3
#> [55] outliers_0.14       glue_1.6.2          purrr_0.3.4
#> [58] processx_3.5.2      pkgload_1.2.4       parallel_4.1.3
#> [61] fastmap_1.1.0       yaml_2.3.5          sessioninfo_1.2.2
#> [64] memoise_2.0.1       knitr_1.37          usethis_2.1.5
```