High-throughput Alternating Least Squares (ALS) with the "alsace" package

Ron Wehrens

October 17, 2016

1 Introduction

Multivariate Curve Resolution (MCR) is a suite of methods aiming to decompose mixtures of signals into individual components resembling true chemical entities. Examples include spectral data measured over time by equipment such as HPLC-DAD: at any point in time the response is a spectrum, and at any wavelength the response is a elution profile. Measuring one sample typically leads to a data matrix, and several samples to a data cube with the temporal and spectral axes in common. Several different algorithms exist, but the most often used one is Alternating Least Squares (ALS, or MCR-ALS), employing Beer's Law iteratively to obtain spectra and elution profiles of "pure" mixture components. Several excellent reviews are available (**??**).

The underlying assumption is that the spectral response is linearly proportional to the concentration of a chemical compound in a mixture (Beer's Law), given wavelength-dependent absorbances:

$$\boldsymbol{X} = \boldsymbol{C}\boldsymbol{S}^T + \boldsymbol{E} \tag{1}$$

Here, $\boldsymbol{X}$ is the measured data matrix, $\boldsymbol{C}$ is the matrix containing the "pure" concentration profiles, $\boldsymbol{S}$ contains the "pure" spectra and $\boldsymbol{E}$ is an error matrix. Suppose we would know the pure spectra, it would be easy, given the experimental data, to calculate the corresponding concentration profiles:

$$\hat{\boldsymbol{C}} = \boldsymbol{X}\boldsymbol{S}(\boldsymbol{S}^T\boldsymbol{S})^{-1} = \boldsymbol{X}\boldsymbol{S}^+ \tag{2}$$

If, on the other hand, we would know the concentration profiles, we could estimate the corresponding spectra:

$$\hat{\boldsymbol{S}} = \boldsymbol{X}^T\boldsymbol{C}(\boldsymbol{C}^T\boldsymbol{C})^{-1} = \boldsymbol{X}^T\boldsymbol{C}^+ \tag{3}$$

MCR-ALS operates by alternating Equations **??** and **??** until the estimates converge, or a maximum number of iterations has been reached. In most applications, one chooses to normalize the spectra in the columns of matrix $\boldsymbol{S}$ to unit length, so that the time profiles can be directly interpreted as relative concentrations.

For R, the basic machinery to analyse such data is available in package **ALS** (**?**) – however, using the approach in practical applications requires quite a lot of additional scripting. This is exactly what **alsace** (**?**) aims to provide for the analysis of HPLC-DAD data, especially when applied in a high-throughput context.

2 Example data

Package **alsace** comes with example data, a very compressed part of the data that have been described in the analysis of carotenoids in grape sample (**?**). By their very nature, these data are quite big and therefore it is impossible to include data from all samples in the package itself. In the near future, however, these will be uploaded to an open data repository so that users can observe the application of **alsace** to the full data set.

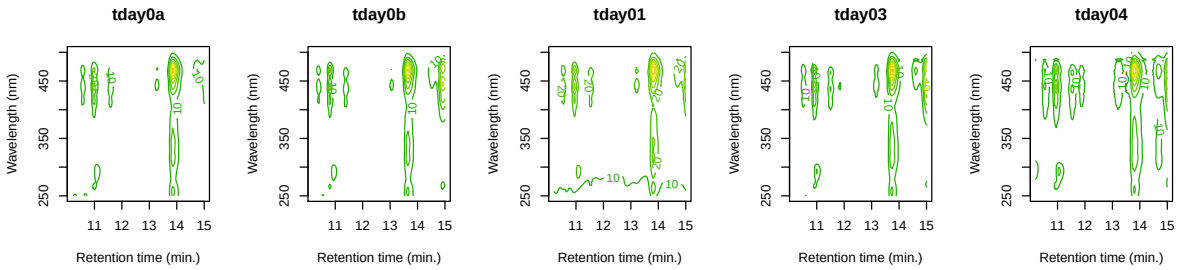

Figure 1: Contour plots of the `tea` data coming with the **alsace** package.

The samples that *are* included in the package are five real grape samples, measured after 0, 0, 1, 3 and 4 days of storage, respectively. The name of the data set is `tea`, which does not refer to the hot beverage but to tri-ethylamine (TEA), used as a conserving agent in these samples. The original publication investigated whether addition of TEA was useful (it was). Each sample leads to a data matrix of 97 time points and 209 wavelengths. Contour plots of the six files are shown in Figure **??**.

It should be stressed that the examples in this vignette in no way should be taken as the ideal or even a reasonably thorough analysis of these data: one disadvantage of ALS is that there is rotational ambiguity, meaning that there are several and in some cases many equivalent solutions. Imposing constraints, such as non-negativity, helps, but may not be sufficient. In general, one can expect better results with larger data sets: including as many samples as possible will only improve results. If some of these samples contain mixtures of known compounds, or are measurements of pure standards, this will signficantly help the analysis in two ways: firstly, rotational ambiguity will be decreased when a few unambiguous samples are present. And secondly, one can use prior information for the initialization: if spectra of compounds that are known to be present are available, this provides a good starting point for the ALS iterations (see below).

3 Basic workflow

The basic workflow of an ALS analysis runs as follows:

1. load the data;
2. do preprocessing;
3. determine the number of components;
4. set up initial guesses for either spectra or concentration profiles;
5. run the ALS algorithm, interpret the results, add, remove or combine components, change settings, and rerun;
6. generate a peak table.

A graphical flowchart is shown in Figure **??**. Each of the steps in this scheme will be discussed briefly below.

3.1 Data loading

Most probably, the data coming from the spectrometer need to be converted into an R-accessible format such as a `csv` file. Once all `csv` files are gathered in one directory (say, `foo`), an easy way to load them is to use `lapply`:

```
> allf <- list.files("foo", pattern = "csv", full.names = TRUE)
> mydata <- lapply(allf, read.csv)
> names(mydata) <- paste("f", 1:length(allf), sep = "")
```

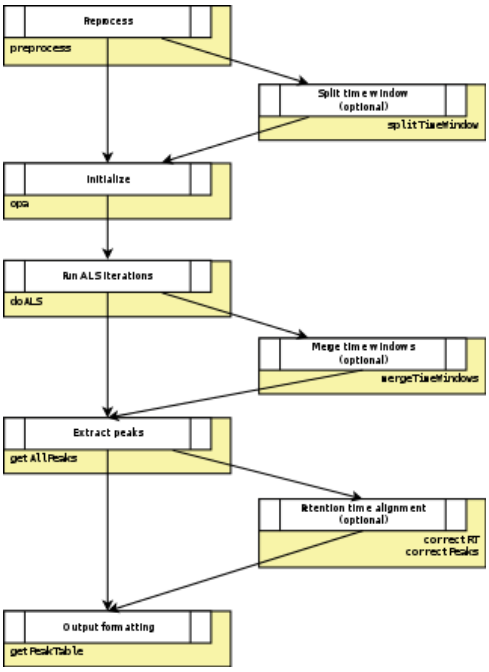

Figure 2: Flowchart of the alsace functionality. The input is a list of data matrices, and the output a table giving peak intensities for all features across the samples.

This will lead to a list of matrices, each one corresponding to one data file – look at the `tea` object to see exactly what is meant. Probably it is a good idea to use more meaningful names than only the number of the file, as is done here. For later ease of interpretation and making plots, it is advisable to put the time points and wavelengths as row names and column names, respectively. The result should look something like this:

```
> lapply(tea.raw[1:2], function(x) x[1:4, 1:6])

$tday0a
          250.4     251.6     252.8      254    255.2    256.4
10.2   8.136870  7.905016  7.653851 7.425400 7.191032 6.979917
10.25 10.302084  9.979436  9.653448 9.351343 9.037995 8.750420
10.3  10.866556 10.502097 10.141724 9.790492 9.436725 9.111691
10.35  9.944014  9.596628  9.241731 8.899640 8.559782 8.252433

$tday0b
         250.4    251.6    252.8      254    255.2    256.4
10.2  8.336324 7.977432 7.609905 7.268445 6.911337 6.585104
10.25 7.319625 7.033055 6.740148 6.477661 6.196796 5.941862
10.3  6.639692 6.451501 6.258161 6.107995 5.922349 5.760366
10.35 6.456141 6.370702 6.281526 6.225763 6.155957 6.072675
```

3.2 Preprocessing

Experimental data can suffer from a number of non-informative defects, such as noise (random or non-random), a baseline, missing values, and many others. In addition, the measurement instrument may provide more data than we are actually interested in: the wavelength range or time range may be larger than we need. HPLC-DAD data are quite smooth in nature, which makes it relatively easy to tackle

these issues: smoothing can remove much of the noise in the spectra direction, baseline subtraction can remove a baseline in the elution direction, and through selection and interpolation we can choose the time and wavelength ranges and resolutions.

All this is available through function `preprocess`. By default smoothing and baseline subtraction are enabled, and providing alternative sets of time points and/or wavelengths will lead to a subsampling and subsequent interpolation both one or both axes. This can signficantly reduce the data size and computational load in later steps. Consider, e.g., doing some preprocessing on the `tea.raw` data:

```
> new.lambdas <- seq(260, 500, by = 2)
> tea <- lapply(tea.raw,
+               preprocess,
+               dim2 = new.lambdas)
> dim(tea.raw[[1]])

[1]  97 209

> dim(tea[[1]])

[1]  97 121
```

The preprocessing by default does baseline subtraction in the temporal direction, and smoothing in the spectral direction. This example also decreases the spectral resolution. It is also possible to set the maximum intensity to a prespecified value: this can be useful when data of chemical standards are analysed together with real samples. In such a case one might give the chemical standards a high weight (by scaling them to a high intensity) – this will help ALS in obtaining meaningful components.

3.3 Estimating the number of components

Estimating the number of components is a difficult business. There are some criteria, but you will never be sure until after you finish your analysis. Currently, **alsace** does not support any automatic choice for picking the right number of components, but it *does* offer the possibility to assess component redundancy, and remove or combine components (see below). In that sense, it is probably best to start with a generous estimate, see which components make sense and which don't, and to refit the model only with the useful components.

3.4 Obtain an initial guess

For HPLC-DAD data, the usual approach is to find time points at which the measured spectrum is as "pure" as possible, *i.e.*, is most likely to come from only one chemical compound. Many different methods have been proposed in literature; **alsace** currently provides the Orthogonal Projection Approach (OPA) (**?**) through function `opa` (**?**). The function returns a set of spectra that is as different as possible. The criterion is based on the determinant of the matrix of spectra selected so far: the one spectrum that, when added to the matrix, leads to the biggest increase in the determinant is considered to be the one that is least similar to the ones previously selected, and will be added. This step is repeated until the desired number of components is reached; the very first spectrum is the one that is most different from the mean spectrum.

The R syntax is very simple indeed:

```
> tea.opa <- opa(tea, 4)
```

This leads to the spectra shown in Figure **??**. The first one, shown in black, does not immediately seem very interesting, but the other three are: the red and blue lines show the typical three-peak structure of carotenoids, while the green line corresponds to the spectrum of tocopherol (**?**).

If we have prior information, e.g., in the form of spectra of compounds known to be present, this can be presented to opa using the `initXref` argument. The function will then add components (as dissimilar to the presented spectra as possible) until the required number of components has been reached.

```
> matplot(new.lambdas, tea.opa, lty = 1, ylab = "Response",
+         xlab = expression(lambda), type = "l")
> legend("topright", legend = paste("Comp.", 1:4), bty = "n",
+        lty = 1, col = 1:4)
```

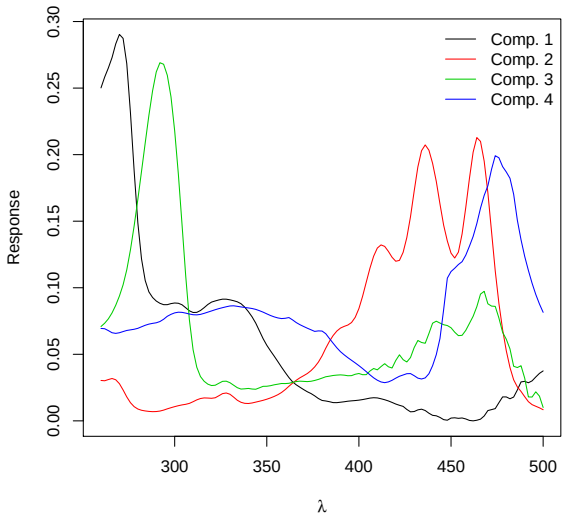

Figure 3: Result of applying OPA to the preprocessed `tea` data: four components are shown. The code to generate the figure is shown on top.

3.5 Run ALS

Once we have initial guesses of the spectra of pure components, it is time to start the ALS iterations. Function `doALS` basically is a wrapper for the `als` function from the **ALS** package with some predefined choices:

- we start from estimated pure spectra, such as given by the `opa` function;
- spectra are normalized to a length of one – this means concentration profiles are not normalized;
- we do not allow negative values in either spectra or concentration profiles;
- we do not fit a separate baseline, since we assume the data have been preprocessed in an appropriate way;
- since in the analysis of natural products it can easily happen that different compounds have the same chromophore (and therefore the same UV spectrum), we do *not* enforce unimodality of the concentration profiles: more than one peak can occur in the elution profile of one particular component.

The result is an object of class "`ALS`", for which `summary`, `print` and `plot` methods are available.

For our small set of example data, this leads to the following:

```
> tea.als <- doALS(tea, tea.opa)
> summary(tea.als)

ALS object fitting 5 samples with 4 components.
Each data matrix contains 121 wavelengths and 97 timepoints
        RMS fit error: 0.44575
```

```
> par(mfrow = c(1,2))
> plot(tea.als)
> plot(tea.als, what = "profiles", mat.idx = 1)
> legend("topleft", legend = paste("Comp.", 1:4), bty = "n",
+        lty = 1, col = 1:4)
```

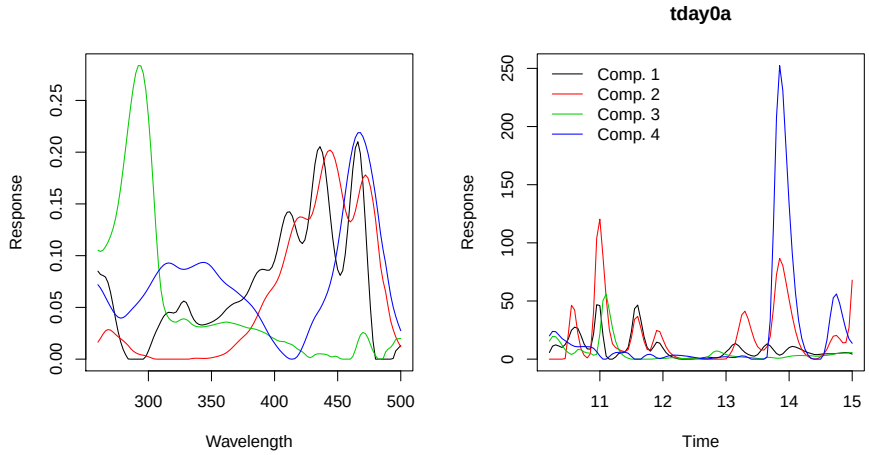

Figure 4: Examples of the generic `plot` method for an `ALS` object: the left plot shows the spectra, and the right plot shows the concentration profiles in the first of the five `tea` samples. The code to generate the plots is shown at the top.

```
LOF: 5.92%
R2: 0.99649
```

Note that the standard output of the underlying `ALS` function, showing for the individual iterations the residual sum of squares, and at the end printing the ratio of the initial versus the final values, is suppressed in `doALS`. Instead, the `summary` can be used after the fitting to obtain more information on the model, presenting three common measures of fit.

The `plot` method for an `ALS` object can show two things: the spectra, and the concentration profiles. Figure **??** shows both, where for space-saving reasons only the concentration profiles of the first file are shown. Comparing the spectra after application of ALS with the OPA results, we see that especially Component 1 has changed. The right plot, containing the concentration profiles in the first file, looks not very smooth due to the low number of time points. Nevertheless, we can see several clear peaks for the carotenoids (blue and red lines), and one clear peak for tocopherol at approximately 11.1 minutes. Notice that around this retention time also two carotenoids elute – ALS is able to resolve such overlap situations because of the difference in spectral characteristics of the individual components.

3.6 Evaluate and modify ALS models

Only in rare cases wil an ALS model be immediately "correct", or even interpretable – where the word "interpretable" means that the spectra correspond with plausible spectra of plausible chemical compounds, and elution profiles show realistic peak shapes. If the number of components is too large, this may result in strange spectra, or elution profiles with very low intensity values. Components corresponding to such low intensities can be identified by the function `smallComps`:

```
> smallC <- smallComps(tea.als, Ithresh = 10)
> smallC
```

```
$smallComps
named integer(0)

$maxCvalues
Component 1 Component 2 Component 3 Component 4
   57.71669   259.97953    61.69982   275.02651
```

The function returns those components that never reach the intensity threshold, and as extra information also presents maximal intensities for all components in all data files. In this case, all four components are over the threshold so none is suggested for removal. If, however, such a small component is identified, the model can be updated by eliminating it from the spectra using function `removeComps`. This will also rerun the ALS iterations. If one does not want to do a complete set of iterations, but only wants to calculate new elution profiles using the reduced set of component spectra, one can add the argument `maxiter = 1`.

One shortcoming of the description of the system by Equation **??** is that it is not unique, the rotational ambiguity already mentioned earlier. One can add an orthonormal rotation matrix $\boldsymbol{G}$ to the equation in the following way:

$$\boldsymbol{X} = \boldsymbol{C}\boldsymbol{S}^T + \boldsymbol{E} = \boldsymbol{C}\boldsymbol{G}^T\boldsymbol{G}\boldsymbol{S}^T + \boldsymbol{E} = \boldsymbol{C}'\boldsymbol{S}'^T + \boldsymbol{E} \tag{4}$$

and the result is exactly the same. There is no way to determine which set of spectra and elution profiles, $\boldsymbol{C}$ and $\boldsymbol{S}$ or $\boldsymbol{C}'$ and $\boldsymbol{S}'$, is "correct", at least not on mathematical grounds.

Visualizing the results, however, may show some of these difficulties. It often happens that the spectrum of one chemical compound is a weighted sum of two of the components. This is most clear when looking at the chromatograms in several samples: if two components show very high correlations at certain peaks (not necessarily all peaks), then these two components could describe the same chemical species. Also in the full version of the data used in this vignette, such behaviour is observed: different carotenoids have very similar spectra, sometimes slightly shifted in wavelength. In some ALS models this leads to one component describing the basic carotenoid spectrum, and another component resembling a second derivative and basically constituting a shift operator. One function that can help in finding such combinations is `suggestCompCombis`; function `combineComps` then allows the user to define which components can be combined, where an original component may be present in more than one new component. For the small data set described in this vignette no really meaningful example can be given. Please refer to the manual pages of these functions, and to the R script describing the application of **alsace** to the complete TEA/noTEA data set.

3.7 Generate a peak table

After the ALS step, we still are some distance away from having a peak table summarizing, for each component, what peaks are found, including information such as retention time and peak area. Such a table is the goal of most metabolomics analyses, and can serve as input for further multivariate analysis. A complicating factor is that retention times can vary, sometimes significantly. This necessitates an alignment step, that puts the same chemical compounds at the same retention time. Application of ALS can greatly help in this respect, since we now have peaks with distinct spectral characteristics that should be aligned. If in all files only one tocopherol peak is found, such as in the right plot in Figure **??**, this already gives us a strong handle to apply the correct retention time correction.

3.7.1 Peak fitting

The first step in generating a peak table is to identify peaks. This is done by the **alsace** function `getAllPeaks`. It works on the list of concentration profiles (one list element for each data file), and takes one other argument indicating the span, i.e. the width of the interval in which local maxima are to be found. A wider span leads to fewer peaks. Note that the `span` parameter is given in terms of number of points, and not in the retention time scale.

For the `tea` data, the function leads to the following result:

```
> pks <- getAllPeaks(tea.als$CList, span = 5)
> sapply(pks, function(x) sapply(x, nrow))
```

```
            tday0a tday0b tday01 tday03 tday04
Component 1     11      9     10     10     10
Component 2      8      9      8      8      8
Component 3      6      8      7      8      7
Component 4      8      8      8      7      8
```

There is clearly some variation: not all peaks are found in all files. Again, it should be stressed that these are low-resolution data, only used for illustration purposes.

3.7.2 Peak alignment

Many retention time correction (or time warping) algorithms have been described in literature. Here we use Parametric Time Warping (PTW) (**?**), as implemented in package **ptw** (**?**). The advantages of this approach include speed, simplicity, and an explicit control over the complexity of the warping function. The last property is essential in preventing false matches to be made. The gist of the `ptw` method is that the time axis is warped with a polynomial function: the higher the degree of the polynomial, the more wriggly the transformed time axis can become. Simply adding a shift or a stretch of the time axis can already be obtained by using a linear function. Note that alignment or time warping is not necessary for the application of ALS itself – on the contrary, time warping of a signal that is already deconvoluted by ALS is much, much simpler than doing it on the original data.

Finding the optimal warping coefficients for a polynomial of a certain degree can be done with function `correctRT`. It takes the list of concentration profiles, and the index of the data set that is considered to be the "reference". Choosing the optimal reference is not a trivial task: in order to make the data distortions by warping as small as possible one could consider choosing a sample somewhere in the middle of the measurement sequence, but also other considerations apply (**?**). In the current example, the choice does not matter much, and we will simply use the very first sample as a reference.

```
> warping.models <- correctRT(tea.als$CList, reference = 1,
+                             what = "models")
```

This will lead to one global warping function for every sample, a function that we can apply to the retention times of the peaks that we identified earlier:

```
> pks.corrected <- correctPeaks(pks, warping.models)
> pks.corrected[[2]][[1]]
```

```
         rt         sd FWHM    height       area   rt.cor
 [1,] 10.45 0.10616523 0.25 28.591869 152.175506 10.41820
 [2,] 10.80 0.02123305 0.05 56.175732  59.797214 10.72758
 [3,] 11.40 0.06369914 0.15 47.543869 151.826642 11.26849
 [4,] 11.75 0.04246609 0.10 15.628845  33.272780 11.59018
 [5,] 12.50 0.06369914 0.15  1.872983   5.981187 12.29479
 [6,] 12.95 0.06369914 0.15 13.545028  43.254705 12.72756
 [7,] 13.45 0.06369914 0.15 10.735290  34.282086 13.21720
 [8,] 13.80 0.12739827 0.30 12.060791  77.029886 13.56547
 [9,] 14.65 0.16986436 0.40  4.353176  37.070497 14.43014
```

The result is stored as an additional column in the peaktable, labelled "`rt.corr`".

3.7.3 Grouping peaks across samples

Once the peaks are aligned, one can think of grouping features across samples – the result can then be presented in the form of a simple matrix, giving peak areas or peak heights for each feature over the whole set of data matrices. Function `getPeakTable` tackles this by performing a complete linkage clustering over the (corrected) retention times for each component. If two peaks from the same sample end up in the same cluster, a warning message is given: in such a case only the most intense response will be included in the data matrix.

```
> pkTab <- getPeakTable(pks.corrected, response = "height")
> head(pkTab)
```

```
     Component Peak       RT   tday0a   tday0b   tday01   tday03   tday04
[1,]         1    1 10.35910 12.27907 28.59187  0.00000  0.00000  0.00000
[2,]         1    2 10.56538 27.53632  0.00000 25.95596 29.78303 30.30119
[3,]         1    3 10.72758  0.00000 56.17573  0.00000  0.00000  0.00000
[4,]         1    4 10.91900 46.79842  0.00000 48.74568 57.71669 55.97018
[5,]         1    5 11.34759  0.00000 47.54387  0.00000 54.86007  0.00000
[6,]         1    6 11.55938 46.50832 15.62885 43.15456  0.00000 48.93419
```

Note that the function also gives graphical output when the argument `plotIt = TRUE` is provided; this may help in identifying problems in the alignment. Again, examples of the use of this feature can be found in the R script for the analysis of the complete TEA data set.

The first three columns of the output of `pkTable` describe the feature: intensities, either given as peak height (as in the example above) or as peak area, are in the further columns. Since the unimodality constraint from MCR-ALS is not applied, we are able to find several chemical compounds belonging to the same ALS component, i.e., with the same spectrum. Especially in the analysis of natural samples this can occur frequently: the chromophores of the compounds, defining the UV-Vis spectrum are the same, but small changes in chemical structure elsewhere lead to different retention behaviour.

4 Useful additional tools

The data processing pipeline discussed so far represents the basic approach to high-throughput analysis of HPLC-DAD data, when careful manual tuning of ALS settings and interactive analysis are practically impossible. This places all the more stress on the post-hoc validation of the models. Visualization is of prime importance. The first topic discussed here shows a way to visualize raw data, fitted values and residuals for several files at once.

Furthermore, one should realize that the computational demands of ALS, when applied thoughtlessly to a large data set with many samples, time points and wavelengths, can be prohibitive: the computer memory needed for the inversion of these large matrices will be quickly impractical. The obvious solution is to chop up the problem in smaller chunks, *viz.* partition the chromatograms in several smaller time windows. Package **alsace** provides facilities to do this, uniquely also offering the possibility to define overlapping time windows. Whereas this may seem inefficient and an unnecessary increase in computational demands, it provides the extremely useful possibility to afterwards merge components that occur in more than one time window – this again leads to one single model for all data combined.

4.1 More elaborate plots of ALS objects

Simultaneous visualization of data matrices, be they experimental data, fitted values, or residuals, is provided by function `showALSresult`. In addition to the data, also the elution profiles and spectra of the fitted ALS components may be visualized. As an example, consider the residual plots of the first two TEA samples, shown in Figure **??**. Elution profiles are shown above the images of the residuals, and spectral components to the right. As can be seen, the biggest residuals are found exactly at the positions of the features. This is a general characteristic of ALS models, and in itself not a source of concern: in

```
> maxResid <- max(abs(range(tea.als$resid)))
> showALSresult(tea.als, tea.als$resid, tp = tpoints, wl = new.lambdas,
+               logsc = FALSE, img.col = cm.colors(9),
+               mat.idx = 1:2, zlim =c(-maxResid, maxResid))
```

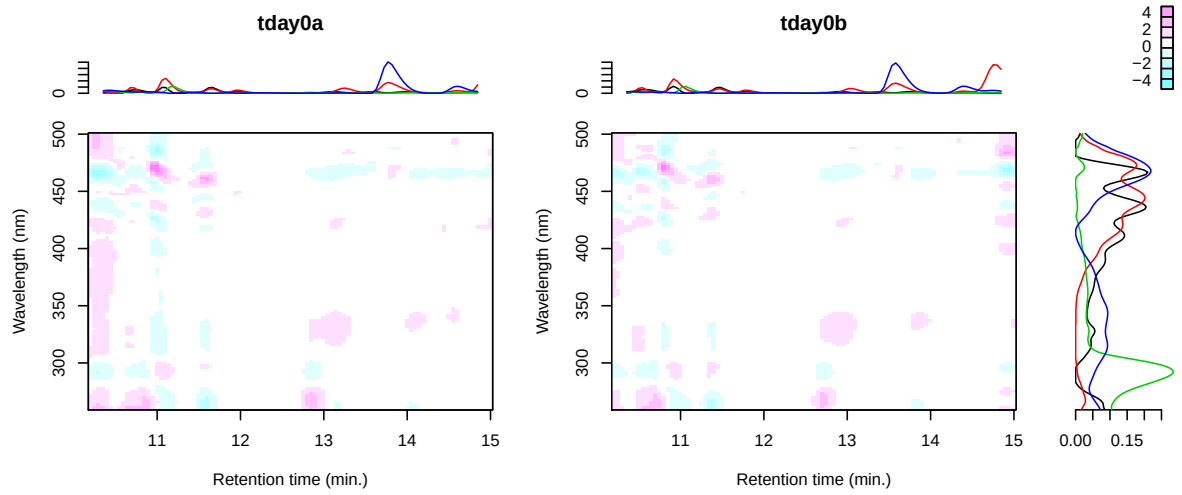

Figure 5: Residual plots for the four-component ALS model on the first two matrices of the `tea` data. The code to generate the plot is shown at the top.

areas where there is no signal the residuals will always be small. Compared to the overal scale of the data in `tea` (the largest values are around 30) the errors are still quite small. Furthermore, it should be bourne in mind that these are demonstration data of low quality. The fit will improve when more data files are added, and when the resolution (especially of the time axis) is increased.

4.2 Splitting and merging time windows

For large data sets, it is imperative to use a divide-and-conquer strategy: memory requirements would otherwise prove too much, even on modern-day computers. Function `splitTimeWindow` allows the user to chop up the time axis in several smaller chunks. Some overlap between time windows may be advisable, to make it more easy to stitch the results for individual windows back together:

```
> tea.split <- splitTimeWindow(tea, c(12, 14), overlap = 10)
```

Next, we can apply `ALS` to each of the time windows, and merge the results back together. Apparently one of the components extends over more than one window. Note that there is no real reason to use the same number of components in the different time windows; indeed, very often the individual time windows are not of the same complexity.

The concentration profiles of each file can be shown using function `plot.ALS`, as discussed before; the difference is that in addition the window borders and overlap areas are shown with gray vertical lines. The result for the `tea.merged` data is shown in Figure **??**. We should have a closer look at this: the two components corresponding to the main peak just before 14 minutes seem to be *very* highly correlated, also in the second peak around 14.6 minutes (at least for the second and fourth files).

For small data files, like the ones used in these examples, the difference in speed is not all that big:

```
> full.time <- system.time({
+   tea.opa <- opa(tea, 4)
```

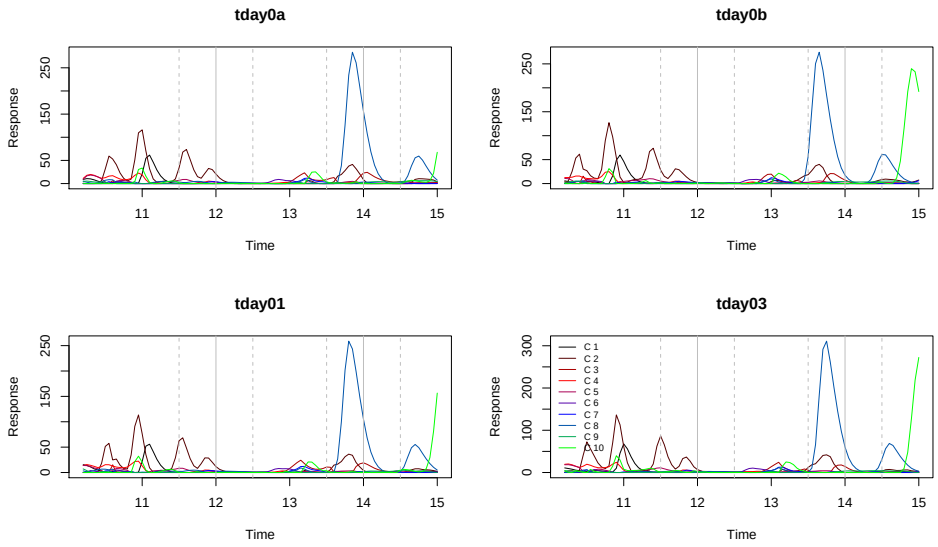

Figure 6: Concentration profiles (`tea` data), obtained after merging the ALS results of the individual time windows. Window border and overlap areas are shown with gray vertical lines.

```
+   tea.als <- doALS(tea, tea.opa)
+ })

> windows.time <- system.time({
+   tea.split <- splitTimeWindow(tea, c(12, 14), overlap = 10)
+   tea.alslist <- lapply(tea.split,
+                         function(Xl) {
+                           Xl.opa <- opa(Xl, 4)
+                           doALS(Xl, Xl.opa)
+                         })
+   tea.merged <- mergeTimeWindows(tea.alslist)
+ })

> full.time

   user  system elapsed
  3.916   0.152   4.068

> windows.time

   user  system elapsed
  3.539   0.111   3.651
```

The difference here is in the order of 15%; for bigger systems, however, time gains will be much more substantial. Note that on POSIX systems like MacOS and Linux easy parallellization can be obtained using the `mclapply` function instead of the normal `lapply` in the previous code fragment.

5 Reference data sets

Currently, upload of the raw data files using the data from **?** and **?** to the metabolights repository (`http://www.ebi.ac.uk/metabolights`) is in progress. Once the necessary database structures have

been defined it will be possible to obtain the ascii data files and scripts that perform the analysis in the two papers mentioned, using **alsace**.