A Brief Tutorial on *batchCorr*:

An R package for between- and within-batch drift correction of high-resolution mass spectrometry-based data

Carl Brunius, Uppsala, 17 October 2018

# What is *batchCorr*?

The *batchCorr* package is a collection of algorithms for correction and cleanup of raw instrument MS data. It was originally developed for untargeted high-resolution LC-MS metabolomics data, but has also found successful application for other MS front-ends (GC and CE) as well as different MS technologies (TOF, QTOF, OrbiTrap, SQD, TQD).

Modern high-resolution mass spectrometers are fantastic instruments for untargeted molecular research, since they allow for a wide coverage of the analytical space, e.g. the plasma, urine or CSF metabolome. However, these instruments are also inherently unstable in their response over time, e.g. due to build-up of dirt in the interface of an LC-MS system or imperfect column regeneration in the gradient program. This lack of stability can affect both the measured m/z, retention time and signal intensity and several approaches have been developed and used to manage this situation. In general, systematic signal deviations are larger between batches than within batches. The approach in *batchCorr* has been to separate variability between and within batches to manage them separately.

The package consists of 3 main modules: i) between-batch correspondence/alignment; ii) within-batch intensity drift correction and; iii) between-batch normalization. The main rationale for the first two modules is to work on information aggregated from several samples (module i) or several features (module ii) to improve signal/noise ratio and reduce likelihood of various types of overfitting.

The package is described in some detail in our paper: Brunius et al Metabolomics 2016 (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5031781/>).

This tutorial describes two frequently encountered use-cases:

* One-batch intensity drift correction. This is the “easy case”, suitable for when you have only one batch of data and want to improve data quality by removing instrumental drift from your raw data.
* Multi-batch between- and within-batch correction: This is the “heavier” case, where we will touch upon correspondence, between- vs within-batch variability, drift correction and normalization.

For both cases, this tutorial will discuss issues regarding the specific tasks as well as present working code using data shipped with the package. You should thus be able to follow this tutorial and generate data output and figures to check that the algorithm as well as your use of it is correct.

# Installation

We assume that R has been downloaded and installed on your computer (<https://www.r-project.org/>). Furthermore, we recommend to download, install and work in RStudio (<https://www.rstudio.com/>) or another IDE of your choice, which has several advantages over working in “simple” command line R. There are several online resources for learning to work efficiently with R and RStudio (e.g. <https://www.rstudio.com/online-learning/>) and you can use any search engine to find good, freely available material. In this tutorial, R code is shown in red monotype font.

1. Install the release version of ‘devtools’ from CRAN:

install.packages("devtools")

1. Make sure that a working development environment has been properly installed.
   * **Windows**: Install [Rtools](https://cran.r-project.org/bin/windows/Rtools/) (<https://cran.r-project.org/bin/windows/Rtools/>).
   * **Mac**: Install Xcode from the Mac App Store.
   * **Linux**: install the R development package, usually called ‘r-devel’ or ‘r-base-dev’.
2. Install *batchCorr* from Gitlab.

library(devtools)

install\_git("https://gitlab.com/CarlBrunius/batchCorr.git")

# Data

The *batchCorr* algorithms will require peak data in table format (peakTable) with injections in the rows and variables in the columns. You will need a vector of injection numbers for modelling the signal intensity drift. For practical purposes, it will be practical to also have a vector of sample type (with e.g. “QC”, “Reference”, “Sample”, “SST”, etc) from which to efficiently extract samples of different types. For multi-batch corrections, a vector with batch identifier is also required. To effectively manage all this information and ensuring the same sample order between all vectors, it is recommended to work with a meta data frame (metadata) that collects all information about batch, injection number, sample type and sample name and possibly other relevant information. Of note: the order of meta data and peak table needs to be the same! This can be achieved by using rownames() and checking e.g. identical(rownames(peakTable),metadata$sampleName). Relevant information is then extracted from the meta data using the $ operator (e.g. metadata$batch).

A note on peak tables: For drift correction, you will need to have a full peak table, i.e. without missing values. For between-batch alignment/correspondence, we also make use of the peak table before filling any missing peaks. In an xcms workflow, this would correspond to the peak table achieved after reading files, picking peaks, performing retention time correction and grouping peaks into common features between samples (correspondence). The full peak table corresponds to the peak table after additionally filling in missing values (hard filling) and/or imputation of missing values. In our workflow, we perform peak picking by xcms (v ≥3.0), fillPeaks(), followed by an in-house multivariate imputation algorithm for remaining missing values, but your mileage may vary.

# One-batch intensity drift correction

First, we will load the *batchCorr* package and some example data shipped with the package:

library(batchCorr)

data('OneBatchData')

This will load 2 objects:

* *B\_PT* which is the peak table (no missing data, i.e. after fillPeaks and imputation)
* *B\_meta* which contains information on batch (all samples from batch B), sample group (QC or Ref) and inj (number in the injection sequence)

Let’s not worry about the details yet. Let’s just start with some actual drift correction:

batchBCorr <- correctDrift(peakTable = B\_PT, injections = B\_meta$inj, sampleGroups = B\_meta$grp, QCID = 'QC', modelNames = 'VVE', G = 17:22)

You’ll have noticed that this clustering operation takes some time (about 2-3 min on a standard laptop). That’s because *batchCorr* uses the *mclust* package which goes through various combinations of *modelNames* (geometric constraints for the clusters) and *G* (the number of clusters tried). In the meantime, let’s look at the arguments:

* peakTable: samples in rows, features in columns – features need unique names!
* Injections: injection number in sequence
* sampleGroups: Vector (length=nrow(peakTable)) of sample type (e.g. "sample", "QC", "Ref)
* QCID: QC identifier in sampleGroups
* modelNames and G: see below

In your work directory, you should have some pdf plots. Let’s look at the cluster\_BIC plot:

What you’ll see is the sort of “goodness of fit” (BIC) of the feature clustering (higher is better). We normally start by allowing many models but with low resolution in *G* to get a rough overview. You can even permit all models by omitting *modelNames*, but this may take a lot of time. Normally, we run the first round with the multivariate mixture *modelNames* with varying volume, i.e. modelNames = c('VVV','VVE','VEV','VEE','VEI','VVI','VII') and *G* = seq(5,35,by=10), which will take approximately 5 min using the example data on a standard laptop. We then reperform clustering with a subselection of *modelNames* (e.g. modelNames = c('VVE', 'VEE') and higher resolution of *G* (e.g. G = seq(15,25,by=2)) – See plots below.

|  |  |  |
| --- | --- | --- |
|  | → |  |

Final *modelNames* and *G* constraints vary between batches and will depend strongly on the quality of the raw data (due to the randomness of instrument quality and drift). For normal-to-high quality instrument data, the number of clusters usually range between 6-25. For batches with worse quality, *G* may increase to >50. However, we generally don’t think it’s reasonable to allow for so many clusters and therefore normally employ approximately 40 as an upper limit. The general procedure described above should help you to find parameters suiting your data in limited time.

The actual drift correction is performed using QC samples by first modelling how the intensity of QC signals (i.e. features) drift throughout the injection sequence. What makes *batchCorr* different from other drift correction tools, is that it employs the above-mentioned clustering approach to group features together into drift profiles to reduce the likelihood of modelling instrumental noise in drift profiles.

Let’s look now at the next pdf of interest, which is going to be named cluster\_G… You’ll find that the drift of roughly 11000 features is captured by roughly 20 clusters (actual number may vary due to randomness in subsampling), corresponding to different drift patterns. The pdf report will show you these 20 clusters. The top half shows data before correction, where each grey line represents one single QC feature and the black line is the average drift profile for the entire cluster. The bottom half shows the QC data after correction.

The *batchCorr* package comes with two ways of performing drift correction: Either with or without using external reference samples to get an unbiased assessment whether drift correction for each drift cluster should be performed or not. Above, we went through how you can perform cluster-based drift correction using QC samples without the use of external reference samples, which is probably the most frequent situation in most metabolomics labs.

In the pipeline of our metabolomics facilities, we are using study-specific QC samples to monitor and correct for drift. However, we also include long-term QC samples in the injection sequence for 2 main purposes: i) To monitor instrument performance and stability over time and; ii) to perform unbiased assessment of whether drift correction actually improves the data quality – if quality is not improved, then drift correction is not performed for that cluster to avoid introducing bias.

If you have an external reference in your batch, the correctDrift() function is easily tweaked to accommodate this situation. You will find that the function call below is almost identical to the previous one – We have simply added a ‘refID’ argument that describes which samples are reference samples (in this case labelled “Ref”):

batchBCorr <- correctDrift(peakTable = B\_PT, injections = B\_meta$inj, sampleGroups = B\_meta$grp, QCID = 'QC', RefID='Ref', modelNames = 'VVE', G = 17:22)

The output of correctDrift() is stored in the batchBCorr object and relevant information can be extracted using the $ operator, e.g. $actionInfo (to see what happened to each cluster), $testFeatsCorr (to extract drift-corrected data) and $testFeatsFinal (to extract drift-corrected data which pass the criterion that QC CV < limit (default is 0.3)).

The correctDrift() function is a convenience wrapper for several other functions. By examining it: View(correctDrift) you will find that it consists of data pre-processing and a call to another wrapper function (driftWrap()) which, in turn, is a wrapper for four underlying functions which are doing the real work. Check the package documentation for details.

* clust(): Group features together into drift patterns. This is by far the most time consuming step.
* driftCalc(): Calculate drift patterns per identified cluster.
* driftCorr(): Apply cluster-wise correction if QC samples become more similar upon correction.
* cleanVar(): Remove features with QC CV > limit value (defaults to 0.3)

# Multi-batch alignment, correction and normalization

Systematic deviations are normally larger between batches than within batches in the rt domain (and to a lesser extent in the mz domain). One approach to manage these between-batch deviations is to widen the rt (and possibly also mz) bandwidth for correspondence (i.e. matching peaks from different samples to the same feature). Another approach, that we presented in our batchCorr paper in the *Metabolomics* journal (Brunius et al), is to optimize the rt bw settings for within-batch correspondence and later aggregate features between batches. This is done using the alignBatches() function. First, we will load a three-batch experiment:

library(batchCorr)

data('ThreeBatchData')

This will load 3 objects:

* *PTnofill* which is the peak table with missing data, i.e. without fillPeaks and imputation
* *PTfill* which is the filled peak table with no missing data, i.e. after fillPeaks and imputation
* *meta* which contains information on batch (samples from three batches: B, F and H), sample group (QC or Ref) and inj (number in the injection sequence)

Let’s look at the between-batch alignment:

## Perform batch alignment

# Extract peakinfo (i.e. m/z and rt of features)

peakIn <- peakInfo(PT = PTnofill, sep = '@', start = 3) # These column names have 2 leading characters describing LC-MS mode -> start at 3

# Perform multi-batch alignment

alignBat <- alignBatches(peakInfo = peakIn, PeakTabNoFill = PTnofill, PeakTabFilled = PTfill, batches = meta$batch, sampleGroups = meta$grp, selectGroup = 'QC')

# Extract new peak table

PT=alignBat$PTalign

In the example case, you will decrease from 11815 to 11284 features. In other words, 531 features that were previously artificially split between batches have now been combined together. Alignment is performed based on aggregating feature missingness per batch and comparing missingness structure between batches: If two features are close in mz and rt and have complementary missingness/presence between batches, they are then combined. However, if your peak picking is done with broader bw settings, or there is no large systematic drift between batches, then the output of the alignBatches() function will alert you to the fact that no alignment candidates were found and, consequently, that no between-batch alignment was possible.

The next step is to break the master peak table and metadata into batches and perform within-batch intensity drift correction (see one-batch example above for details and explanations, especially *G* and *modelNames*). However, please note that the below code takes several minutes to run.

# Batch B

batchB <- getBatch(peakTable = PT, meta = meta, batch = meta$batch, select = 'B')

BCorr <- correctDrift(peakTable = batchB$peakTable, injections = batchB$meta$inj, sampleGroups = batchB$meta$grp, QCID = 'QC', G = seq(5,35,by=3), modelNames = c('VVE', 'VEE'))

# Batch F

batchF <- getBatch(peakTable = PT, meta = meta, batch = meta$batch, select = 'F')

FCorr <- correctDrift(peakTable = batchF$peakTable, injections = batchF$meta$inj, sampleGroups = batchF$meta$grp, QCID = 'QC', G = seq(5,35,by=3), modelNames = c('VVE', 'VEE'))

# Batch H

batchH <- getBatch(peakTable = PT, meta = meta, batch = meta$batch, select = 'H')

HCorr <- correctDrift(peakTable = batchH$peakTable, injections = batchH$meta$inj, sampleGroups = batchH$meta$grp, QCID = 'QC', G = seq(5,35,by=3),modelNames = c('VVE', 'VEE'))

As discussed in the one-batch example above: If your workflow uses external reference samples, then these can be specified for unbiased estimation of whether correction should be performed (per drift cluster). This is done by adding RefID = 'Ref' to the correctDrift() function call. The above code produces 9239, 7764 and 9395 features that pass the criterion of QC CV <30% for batches B, f and H, respectively.

The final step is now to merge the drift-corrected data and normalize:

mergedData <- mergeBatches(list(BCorr,FCorr,HCorr))

normData <- normalizeBatches(peakTable = mergedData$peakTable, batches = meta$batch, sampleGroup = meta$grp, refGroup = 'Ref', population = 'sample')

PTnorm <- normData$peakTable

First, you’ll notice that executing the 2nd function call will generate an error, since no true samples are shipped with the example data and population = 'sample' will not be able to select any samples. To try out the algorithm with the example data, you can cheat a little and specify population = 'all' instead of population = 'sample'. This will select all the samples within the batch for normalization by population median (see below).

The mergeBatches() function will combine features that after correction pass the criterion of QC CV <30% in at least 50% of the batches, resulting in a peak table of 9142 features. This ratio can be tweaked by changing the qualRatio parameter of the function call (see ?mergeBatches).

The normalizeBatches() function will check if the reference samples pass certain criteria (see Brunius et al. Metabolomics for details) and, if so, normalize by them. Batches where reference samples don’t pass the criteria will be normalized by population median. The actual population is specified by the population argument, which should match a sample type in the sample group vector (e.g. sample). If you wish to select all samples in the batch, including QCs and potentially other samples, you can specify population = 'all'.

Congratulations, you should now be all set to be able to perform both one-batch intensity drift correction as well as a full-blown multi-batch correction using the *batchCorr* approach!

Good luck and thanks for your interest in the *batchCorr* package,

Carl Brunius

PS. Please don’t hesitate to contact me if you experience any issues with the package / functions or if you have feedback on functionality or tutorial.

[carl.brunius@chalmers.se](mailto:carl.brunius@chalmers.se)