Analysis of noisy data from deuteration experiments

Antonios Zagaris, WBVR/WUR

September 26, 2017

1 Summary

1.1 Data descriptives

The original data set consists of a number of .csv files (*datasets*), each of which has precisely 4 columns named (L–R)

```
## [1] "Counts-435" "Counts-436" "Counts-435" "Counts-436"
```

R enforces unique column names—for good reason—and thus interprets these names as

```
## [1] "Counts.435"   "Counts.436"   "Counts.435.1" "Counts.436.1"
```

The datasets are partitioned by *cell type*, with the represented types being

```
## character(0)
```

Each (cell) type is assigned a dedicated subfolder in the folder Data/2017.06.29_Lars, and that subfolder holds the original datasets as communicated by Lars. Each such subfolder contains several datasets corresponding to *distinct animals* but to *the same type*; the dataset–animal correspondence is reported in a file entitled Decode [...].xlsx housed in that same subfolder. Importantly, that same file also reports the correspondence between datasets and the times (in weeks) at which they were collected. The only other file in that subfolder, apart from datasets and the decoding file, is the file Time.xlsx whose significance is explained below.

The numbers 435 and 436, in the column names, represent the *mass–to–charge–ratios* (m/z) of undeuterated and deuterated deoxyribose moieties, respectively. Each of the 4 columns of any given dataset reports integers (*counts*), and each column entry is the number of deoxyribose particles that (a) have the column-specific m/z and (b) were detected over a specific time interval. That time interval can be deduced from the aforementioned Time.xlsx, which consists of a single column that lists time instants $t_1, \ldots, t_N$. Specifically, the $n-$th entry of the $m-$th column, c_{nm}, lists the number of detected moieties in the interval $[t_{n-1}, t_n]$ or $[t_n, t_{n+1}]$. Since the time grid is (mostly) uniform (see below for the occasional exception), the precise nature of the time interval is (largely) irrelevant.

1.2 Objective

The scope of this report is to quantify the relation between *sample deuteration levels* and *sample collection times*. According to the standard model, the relation is piecewise exponential with deuteration levels increasing toward saturation, at first, and decreasing past a certain time instant. Data fitting using this parametric model provides estimates for the salient model parameters, and thus for cellular *turnover rates*. In the datasets this report deals with, the collected samples have a prohibitingly low signal–to–noise (S/N) ratio, a fact that makes data fitting all but nonsensical. In view of that, this report first aims at developing a methodology that can reliably denoisify data and, subsequently, at applying said methodology to Lars's datasets.

1.3 Data preparation

The data in question have undergone a first curation and the results were stored in the folder Results; all subsequent algorithms load and operate on the data specifically stored in that folder. For starters, all datasets

were cropped past a certain cutoff time, so that they all list 2886 distinct time instants; this is done for reasons petaining to how R reads and processes .csv files. The cutoff time is well outside the range containing the signal of interest, hence this does not affect results in any way. Additionally, the files Decode [...].xlsx and Times.xlsx (in each subfolder) were exported to weeks.csv and times.csv, respectively (also in each subfolder). Moreover, the column File_name (in Decode [...].xlsx) was renamed to Filename (in weeks.csv) to comply with naming conventions within R. Note that the length of the column in times.csv does *not* necessarily match that of the datasets (2886), as neither have listed time instants been cropped nor have missing times been added. The length difference is $\sim 0.1\%$; since we are uninterested in the exact correspondence between times and counts, this was deemed negligible and no attempt to reconcile datasets and times was made.

1.4 Data quantification

To denoisify and (subsequently) quantify the data, we follow the procedure outlined below. The algorithm operates separately on each *type*, and within that on each *dataset*; the information on *times* and *weeks* is retrieved, for each dataset, using the *type* the dataset falls under.

1. Set *rough* lower/upper time limits t.limsL/t.limsR that bound below/above the signal's retention times.
2. Load the times t.g for the currrent type, then append missing time instants to make t.g a unifom grid with step dt.g over the time interval [t.limsL , t.limsR].
3. Extract (average, denoised) undeuterated (H_2O) and deuterated (D_2O) profiles for *start and end controls*. (Start/end controls open/close the *single* GC/MS run in which *samples of the current type* have been quantified.) This yields four *pure* profiles: H_2O–start, H_2O–end, D_2O–start and D_2O–end.
4. Identify the two peak locations and inter-peak "*valley*" location (where the inter-peak profile bottoms out) for the pure profiles.
5. Identify upper/lower cutoff times (bounding 95% of the profile AUC between them) for the pure profiles.
6. Define *individual*, H_2O and D_2O cutoff times and "valley" locations *per dataset*.
7. Prepare datasets by restricting to [t.limsL , t.limsR] and interpolating missing values on the grid.
8. Extract the contaminant (*pollutant*) D_2O profile by averaging and denoising *pure pollutant* datasets.
9. Repair D_2O profiles (except controls/pure pollutants) by estimating and removing pollutant component.
10. Quantify D_2O/H_2O ratios using the repaired profiles.

These steps are covered in detail below.

2 Detailed algorithm

Step 1: rough time limits [**settings.R**]

The objective of this step is to *contain* the relevant part of each signal (i.e. dataset) in the *rough* time interval [t.limsL , t.limsR]. The rationale behind it is enabling us to automate the analysis, by excluding irrelevant peaks and valleys that would confuse the code. This is done *once* and applies to *all datasets across all types*, so the lower/upper time limits t.limsL/t.limsR should bound the relevant part of *all* those signals. Under current settings, t.limsL=5.00 and t.limsR=5.25.

Step 2: missing time points [**presets.R**]

The objective of this step is to "repair" the time instants (times) t.g corresponding to the current type and reported in weeks.csv. Although the times in t.g appear to form a grid with timestep dt.g=0.001, certain grid

values (times) are conspicuously missing. Since times are reported down to 3 decimal digits, this could be the combined effect of dt.g slightly exceeding 0.001 and *round-off*. Such a mechanism would make "skipping" (quasi-)periodic, however this is not the case across cell types. For example, see below for the *differences* (skip lengths) between *successive* missing instants for a specific type:

```
## [1] "Jejunum_CD11b"
```

```
##  [1] 71 68  2 69 69 70 70 70 70 70 71 69 70 71 69 70 69 70 70 70 70 70 70
## [24] 70 70 71 69 70 69  2 68 70 70 70 70 70 70 70 70 70 70 71
```

Instead, I consider "missing" times to be *actually missing* and append them to t.g at the appropriate places. This makes (the updated) t.g a unifom grid with step dt.g over [t.limsL , t.limsR]. This step is performed once per type, not per dataset.

Appending (")missing(") times to the grid will, down the line, require a scheme to fill in profile values (counts) at these (formerly) missing points. This rekindles the discussion on the underlying mechanism, as the "repair" scheme depends on it. An alternative hypothesis is that the apparatus doubles, from 0.001 to 0.002 and every so often, the interval over which it *aggreates* counts. Nevertheless, counts do not appear to double over such intervals, and this mechanism must be discredited as well. Fortunately, a mere $\sim 1.5\%$ of time points is missing for this data, so any reasonable scheme will do. Below, I fill in missing counts by interpolating linearly from their nearest neighbors.

Step 3: pure profiles [ctrl.prof.tab.R]

The objective of this step is to construct uncontaminated (*pure*) H_2O and D_2O profiles for the current type. These will be used in two ways. First, to set the time window that contains the signal in the actual (non-control) datasets. And second, further down the pipeline I will assume each available D_2O dataset to be a weighted sum of a *pure* profile and a *contaminant* profile. Upon deducing the weights, I will be able to remove the (estimated) *contaminant* contribution from the dataset, leaving behind *pure* signal on which to base the computation of the H_2O/D_2O ratio. This necessitates *estimates* of the *pure* (here) and *contaminant* (step 8) profiles for the current type.

Pure profiles are constructed (per type) from the controls, with the complication that these are *several* and come in *start/end* varieties. The *start* (*end*) control set is composed of multiple datasets, measured by the apparatus (in duplicate) at the *beginning* (*end*) of the *single* run quantifying *all samples of that cell type*. (If *start* (*end*) controls are missing for some type, I copy them from that type's *end* (*start*) controls.) Pure profiles are constructed by averaging and mollification according to the procedure below.

- Restrict H_2O–start controls to the interval [t.limsL , t.limsR] by deleting entries corresponding to t.g $<$ t.limsL or t.g $>$ t.limsR.
- If there were missing values in the *restricted* t.g (cf. step 2), estimate (in duplicate) the value of each control profile at those missing instants by linear interpolation from its neighbors.
- Normalize these control profiles (independently and in duplicate) so that each has unit AUC.
- Average these controls (including duplicates) to obtain an *average* H_2O–start profile.
- Prepare a symmetric set of positive weights by discretizing a Gaussian kernel. The kernel's standard deviation sgm (2; in units of dt.g) and number of weights (6*sgm+1) are set manually [settings.R]. (An optimal estimate of sgm on the basis of the variance between individual controls is overkill.)
- *Mollify* the average H_2O–start profile using these weights, then renormalize to unit AUC. To avoid artifacts at the rough interval boundary, all control profiles must be effectively zero near it.
- Repeat the steps above for H_2O–end, D_2O–start and D_2O–end profiles.

This step is performed per type, not per dataset, and yields *unique* start/end control profiles for *both* H_2O and D_2O controls. These profiles are tabulated over the regular grid t.limsL , t.limsL + dt.g , ... , t.limsR.

Step 4: pure profile peak/valley locations [**ctrl.lims.tab.R**]

The purpose of this step is to identify the locations of the two peaks and of the single valley between them, for each of the four pure profiles (constructed in step 3). This information will be used in the next steps, where I refine the estimate of the interval containing the signal.

- Define [settings.R] a natural number N.w, then identify all times in t.g that yield profile counts matching or exceeding those of their 2*N.w-many, left and right neighbors. For N.w=1, one would obtain all *local* maxima; however, nearly all of these are induced by noise. The value N.w$\sim \log_4(\varepsilon/\text{N.t})$, with N.t the number of entries in t.g, ensures (under symmetric noise) that spurious maxima are absent with probability $1-\varepsilon$. For $\varepsilon = 0.01$ and the settings here, N.w ~ 10.
- Define [settings.R] a threshold peak.thr, then discard any remaining peaks lower than peak.thr*highest found peak. These are unlikely to be noise-induced and presumably belong to the pure profile, but discarding them in no way affects that profile. For the settings here, peak.thr ~ 0.05.
- Find the valley as the location, between the two peaks, where the profile has a minimum.

This step is also performed per type.

Step 5: refined intervals for pure profiles [**ctrl.lims.tab.R**]

The purpose of this step is to refine the signal-containing intervals for the four pure profiles (constructed in step 3). Using these lower/upper cutoff times, we will then define analogous intervals for (the rest of) the datasets in the next step. The operating principle is to define (per profile) an interval tighter than [t.limsL , t.limsR] by sacrificing 100*peak.thr% of the total AUC (for the settings here, 5%). This entails that 100*peak.thr% of the profile AUC will have to be trimmed from the left and right profile tails.

- Load the H_2O–start pure profile (having unit AUC).
- Distribute peak.thr of the AUC between left/right tails, proportionally to the AUC held by the left/right modes ("*humps*"). Since modes are not well-separated, estimate the ratio of AUCs held by the left/right modes by the ratio of AUCs below/above the left/right peaks.
- Use this estimate to find upper/lower cutoffs. In technical terms, the lower (upper) cutoff is the maximum (minimum) entry of t.g with the property that the profile AUC from t.limsL to it (from it to t.limsR) does not exceed the AUC to–be–discarded allocated to the left (right) mode.
- Repeat the steps above for H_2O–end, D_2O–start and D_2O–end profiles.

This step is also performed per type and yields upper/lower bounds for each of the four pure profiles.

Step 6: refined intervals for individual profiles [**quant.prof.R**]

This step is similar to step 5 but concerns individual profiles. Defining tighter intervals for these is *absolutely necessary*, because the AUC of the D_2O profiles is affected by the substantial noise present over the interval [t.limsL , t.limsR]. Tighter intervals cannot be "eyeballed" because of this very noise, so we base our estimates on the pure profile intervals (constructed in step 5).

- Fix a dataset, then read from weeks.csv the order in which it was processed in the GC/MS run (say as n-th out of N *non-control* datasets).
- Read the lower cutoff times tL.H2O.start/tL.H2O.end for the H_2O–start/end pure profiles (of current type) constructed in step 5.
- Partition the interval [tL.H2O.start , tL.H2O.end] evenly in $N+1$ sub-intervals, tL.H2O.start , tL.H2O.1 , ... , tL.H2O.N , tL.H2O.end.

- Assign tL.H2O.n as lower cutoff time for the current dataset. This assumes, for lack of a viable alternative, that cutoff times are "*sliding*" linearly with time as the GC/MS run progresses.
- Repeat steps above for upper cutoffs (using *upper* cutoff times tR.H2O.start/tR.H2O.end).
- Repeat steps above for D_2O profiles (using the D_2O pure profile cutoff times).

The efficacy of this '*linear interpolation*' scheme can be partly asessesed using the H_2O profiles, since tight intervals for these are easier to "eyeball" because of relatively low noise–to–signal ratios. The results are *not* encouraging: e.g., the profile plots of Ileum_CD21 below show that H_2O profiles do not "slide" (let alone linearly) from *start* to *end* pure profiles. For example, although Ileum_CD21_5 should trace closely the start pure profile (derived using Ileum_CD21_$1-4$), it lies even further from it than the end pure profile does (derived using Ileum_CD21_$23-26$). (Although the left D_2O peak falls squarely on that of the start pure profile, that must be due to *elevated* pollutant presence; note how the D_2O peak ratio is off the charts.) A possibly viable alternative for tight interval calibration may be to maximize the *cross correlation* between the H_2O profile (but not those of the—possibly pollutant-dominated—D_2O one) and a linear combination of the start/end pure profiles, i.e. to allow for an extra shift.

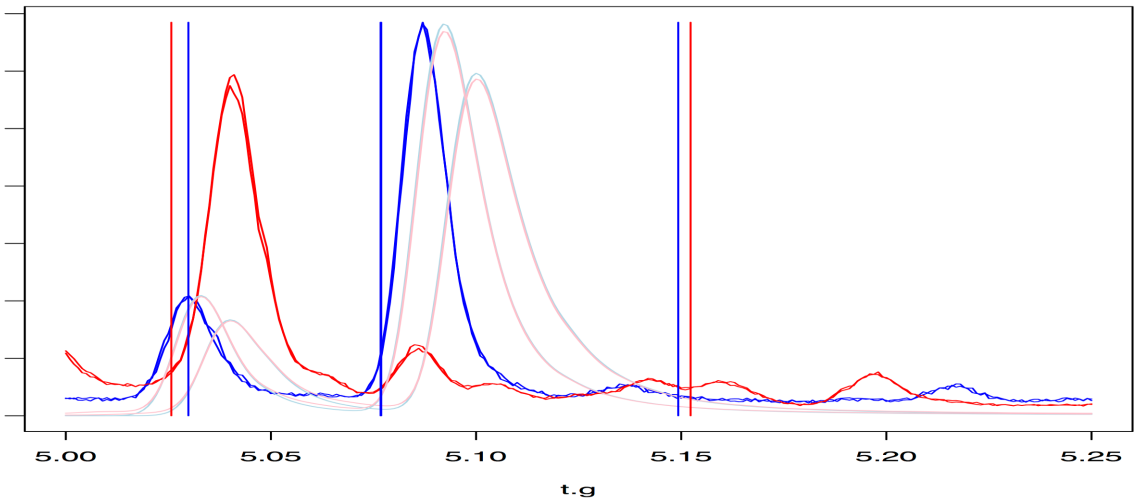

Figure 1: Ileum_CD21 profiles (density over retention time; profiles normalized to unit AUC). The H_2O/D_2O pure profiles are plotted in light blue/red; the rightmost/leftmost profiles correspond to (averaged, mollified) start/end controls. The H_2O/D_2O profiles of Ileum_CD21_5 are plotted in bright blue/red. The corresponding lower cutoff, valley and upper cutoff are plotted as vertical lines, also in bright blue/red.

Step 7: profile preparation [**quant.prof.R**]

This purpose of this step is to prepare datasets for quantification (i.e. for computing their deuteration levels). It yields H_2O/D_2O datasets (in duplicate) that extend over a shorter, uniform time grid.

- Select a dataset, then load its H_2O profiles (duplicate).
- Trim the dataset using the corresponding H_2O lower/upper cutoffs (constructed in step 6).
- Append, at the appropriate places, (any) time instants missing from the grid; estimate missing profile values (counts) by linear interpolation from their immediate neighbors.
- Repeat steps above for the dataset's D_2O profiles.

Step 8: contaminant profile [**poll.prof.R**]

This purpose of this step is to construct a D_2O profile for the contaminant (*pollutant*) corresponding to the current cell type. (There is no evidence of contamination in H_2O profiles.) This and the *pure* profiles act as inputs in step 9, where datasets are split (*decomposed*) into "*signal*" and "*pollutant*" components. The construction traces closely that for pure profiles (step 3), with the salient difference being that there are *no* datasets designated as "pollutant controls" (i.e. *signal-free* or *pure pollutant* profiles). These must be *manually selected*, instead, as profiles that (presumably) hold much more pollutant than signal. To do that, I work with the *peak height ratio*, which remains *approximately constant across controls* but *varies wildly across samples*. Specifically, I declare certain (D_2O) datasets as "*pollutant-only*" if their peak ratio is *severely* skewed (relative to that of the controls). For example, the selected profiles for Ileum_CD21 are the following.

```
##  [1] "Ileum_CD21_7"  "Ileum_CD21_10" "Ileum_CD21_11" "Ileum_CD21_12"
##  [5] "Ileum_CD21_14" "Ileum_CD21_15" "Ileum_CD21_16" "Ileum_CD21_17"
##  [9] "Ileum_CD21_18" "Ileum_CD21_19"
```

Once selected, these profiles are averaged, mollified and normalized over the *rough* interval to yield an unequivocal *contaminant profile* for the current type. This profile is invariably localized (primarily) in the left sub-interval (i.e. left of the valley), meaning that it corresponds to a *well-defined peak*; cf. plot below.

- Declare and store in poll.num [settings.R] *pure pollutants* datasets; selection is done by profile inspection.
- Extract an average, mollified, *pure pollutant* profile with unit AUC, by subjecting the stored pollutant profiles to the procedure detailed in step 3.

This step is performed per type, not per dataset, and yields a *unique* D_2O contaminant profile tabulated over a regular time grid covering the rough interval [t.limsL , t.limsR]. The pollutant profile for Ileum_CD21 is sketched below.

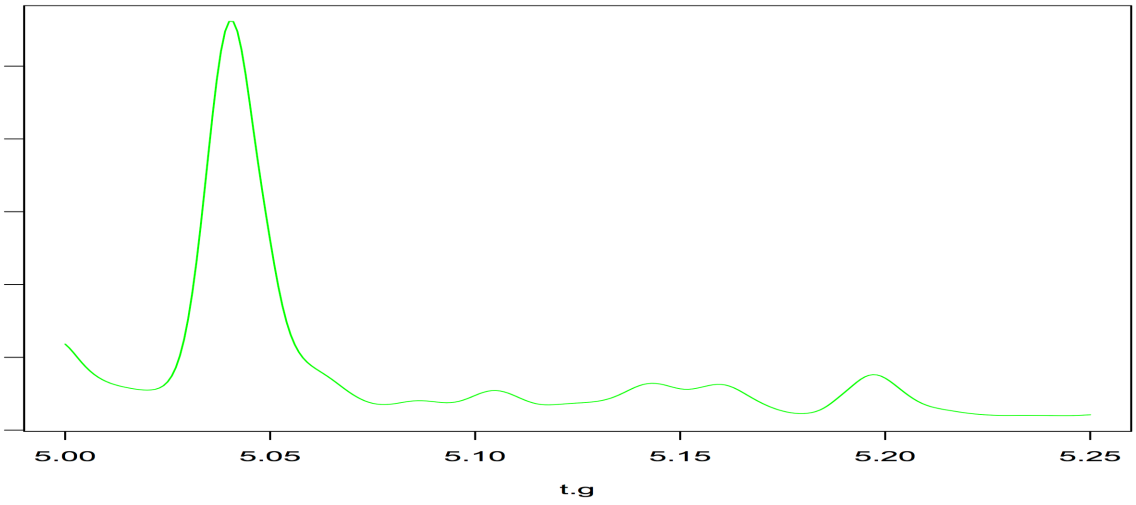

Figure 2: Estimated Ileum_CD21 pollutant profile, nearly exclusively contained in the left sub-interval.

Step 9: profile purification [**quant.fix.R**]

This step is the algorithmic crux of my approach, in that it aims to *repair* D_2O profiles by *removing* from them the contaminant component. The central idea is to write each dataset in the form

$$\mathrm{f_{data}(t)} = \mathrm{a}\,\mathrm{f_{pure}(t)} \;+\; (1-\mathrm{a})\,\mathrm{f_{poll}(t)}, \quad \text{for all } t \text{ in } t.g\,. \tag{1}$$

Here, f_{data}, f_{pure} and f_{poll} are the *profiles* corresponding to the (current) *dataset*, *pure profile* and *contaminant profile*, respectively. For definiteness, we assume all of them normalized to *unit AUC*. Further, $0 \leq a \leq 1$ is a constant (*weight*) quantifying the *contribution* of the *pure* profile to the dataset; $a = 1$ corresponds to pure signal and $a = 0$ to pure contaminant. Such a decomposition is known as a *statistical mixture model*; if a can be *somehow* estimated, then the operative quantity in our analysis can also be estimated via $AUC_{purifieddata} = a\ AUC_{data}$.

There are various ways to estimate a using the information we have. Seen as a literal equation for the *scalar* a, (1) cannot be satisfied for any value of a, because f_{data}, f_{pure} and f_{poll} are linearly independent vectors. Decomposition (1) is instead meant *statistically*, in that it involves *distributions* which we only know imperfectly. Specifically, we only have *estimates* $\hat{f}_{data}$, $\hat{f}_{pure}$ and $\hat{f}_{poll}$; not the distributions themselves. An additional complication is the *approximate* nature of (1), which derives from our simplistic modeling of how f_{pure} amd f_{poll} evolve during the GC/MS run (the former interpolates two profiles, the latter is constant) and therefore raises certain *optimality* concerns (not addressed here).

The normative way of estimating a is by posing it as a *hidden states* problem and treating it numerically with the *EM algorithmic family* [2, 3], implemented in R as mixtools [1]. An "off the shelf" application of mixtools is presumably impossible here, since the pure/contaminant profiles are *tabulated* (not known analyticallly). An expedient solution would be to apply [1] after "modeling" all distributions as Gaussians, but preliminary results in that direction are atrocious. Since the optimization problem is just 1-D (only a need be optimized), a viable alternative would be to develop a "homebrew" (in its simplest form, an exhaustive search). This possibility is worth considering but, in the interst of time, I chose for a simpler option described below.

(works invariably in one direction, elevating the left-over-right peak ratio; importantly, this implies that the pollutant primarily interacts with the left peak, i.e. is largely localized left of the "valley.")

As discussed above, the pollutant profile is largely localized in the left sub-interval and thus affects the left peak more strongly than the right one. Importantly and by the same token, then, pollutant presence alters the *AUC proportion* between *left and right sub-intervals*, hence we can use that proportion to backward engineer the amount of pollutant present in a dataset. To work out a formula, write $t.g.L$, $t.g.V$ and $t.g.R$ for the left cutoff, valley and right cutoff times of a given dataset, respectively. Letting $AUC^{(L)}$ and $AUC^{(R)}$ be the left and right AUCs of an arbitrary profile f, we find

$$AUC^{(L)} = \int_{t.g.L}^{t.g.V} f(t)\,dt \approx \left(\sum_{t=t.g.L}^{t.g.V} f(t) \right) dt.g \quad \text{and} \quad AUC^{(R)} = \int_{t.g.V}^{t.g.R} f(t)\,dt \approx \left(\sum_{t=t.g.V}^{t.g.R} f(t) \right) dt.g.$$

Decomposition (1) implies, then, the AUC decomposition

$$AUC^{(L)}_{data} = a\,AUC^{(L)}_{pure} + (1-a)\,AUC^{(L)}_{poll} \quad \text{and} \quad AUC^{(R)}_{data} = a\,AUC^{(R)}_{pure} + (1-a)\,AUC^{(R)}_{poll}. \tag{2}$$

(Note that the corresponding equation for the entire interval merely yields the identity $1 = 1$.) It is trivial to show that these two equations yield the same coefficient a, using that $AUC^{(L)} + AUC^{(R)} = 1$ for all three AUC types (data, pure and pollutant). This means that either of the left/right intervals can be used to compute that coefficient.

Step 10: profile quantification

Evident (coming up). Note that profiles tagged "pollutant-only" are excluded from the decomposition in step 9, meaning that they cannot be put on the deuteration curve.

References

[1] T Benaglia, D Chauveau, DR Hunter and DS Young (2009), mixtools: An R Package for Analyzing Finite Mixture Models, Journal of Statistical Software 32(6).

[2] M Blume (2002), Expectation maximization: A gentle introduction.

[3] AP Dempster, NM Laird and DB Rubin (1977), Maximum likelihood from incomplete data via the EM algorithm, J the Roy Stat Soc B 39(1):1–38.