Protocol for evidence-based normalization

Brian Low, Tao Huan*

Version 1.0 (2024-07-18)

Overview

For a fair comparison between biological samples in LC-MS-based untargeted metabolomics, sample normalization is needed. This step is critical to minimize sample-to-sample variation. Here, we focus on post-acquisition sample normalization, in which the metabolic intensities are adjusted after LC-MS analysis. The performance of post-acquisition sample normalization is significantly affected by the choice of normalization algorithm, data structure, and data quality. Here, we present an evidence-based normalization workflow to ensure optimal normalization results and meaningful biological insights. Detailed information on each step will be elaborated below.

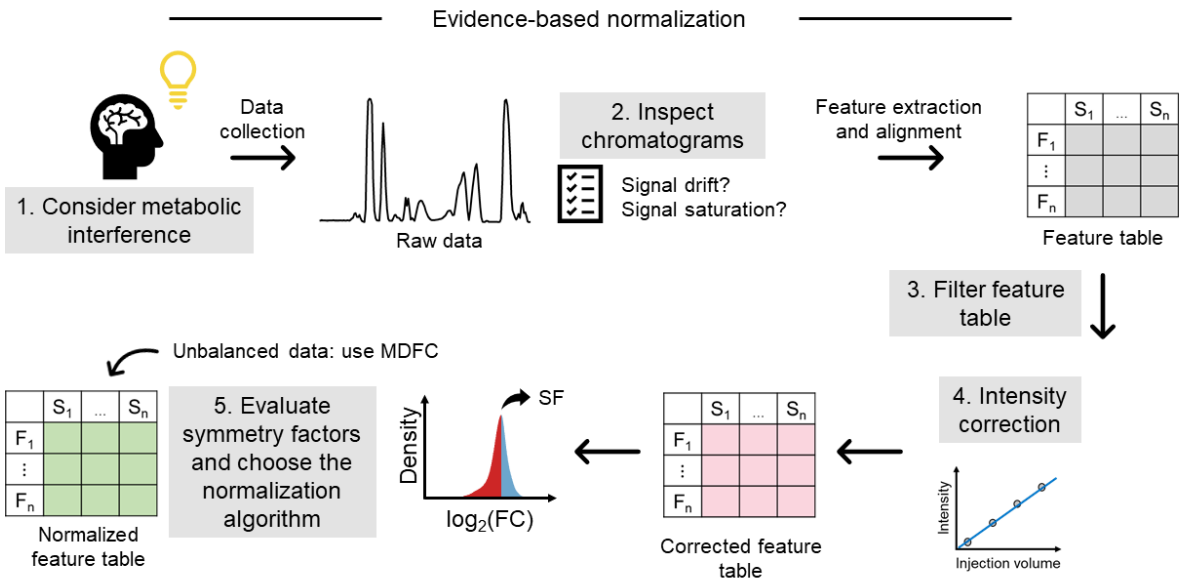


Figure 1: Schematic workflow for evidence-based normalization.

1. Consider metabolic interference

In the experimental design and prior to data acquisition, it is important to carefully check for metabolic interference from sample contamination. Metabolic interference can severely compromise metabolome analyses. For example, in an untargeted study on the saliva metabolome, residual food particles may be present in the saliva samples. These contaminating food particles can introduce additional metabolites and/or artificially

increase the concentration of common metabolites between the food particles and saliva samples. In addition, contaminating metabolites from food particles can interfere with the quantification of the saliva metabolome due to additional matrix effects. Thus, during experimental planning, it is critical for researchers to be aware of any possible sources of contamination so that strict sample handling procedures can be implemented. They should also conduct careful visual inspections on the collected samples. For example, if a saliva sample is not clear and food particles are seen, then the researcher should ask the individual to thoroughly rinse their mouth before providing another saliva sample.

2. Data collection

During data acquisition, method blank (MB) and quality control (QC) samples should be analyzed. MB samples are prepared the same way as the biological samples but without any analyte. These samples allow users to recognize non-meaningful features that arise during sample preparation and data acquisition. In contrast, QC samples are prepared by aliquoting an equal volume from each individual sample. Since QC samples are similar in composition to the tested samples, they can be used to monitor instrument stability to ensure reproducible data. Notably, QC samples should also be injected at increasing volumes (serial QC samples). The role of MB, QC, and serial QC samples will be further elaborated in the following data preprocessing steps.

3. Inspect chromatograms

Prior to data preprocessing, chromatograms should be manually inspected for data quality. First, each sample should be checked to make sure there is adequate signal and minimal signal saturation. If this is not the case, users need to adjust the injection volume and re-collect the LC-MS data. The QC samples can also be used to evaluate instrumental stability. Since the QC sample is repeatedly injected throughout the LC-MS sequence, the MS signals should theoretically be the same. Thus, the QC sample chromatograms can be overlaid and manually inspected. If all the chromatograms overlap each other well, this is indicative of minimal signal and retention time (RT) drift. Otherwise, users need to address instrumental drift. A list of methods is provided and discussed in the following review article: https://doi.org/10.1016/j.trac.2023.117009.

4. Feature extraction

After chromatogram inspection, users can perform feature extraction using either vendor provided or open-source software. Users then need to format their feature table to one compatible with our following workflow as shown here:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Alignmen	RT	HY_1_P1.(	HY_2_P1.(	HY_3_P1.(	HY_4_P1.(	HY_5_P1.(	MB_P1.B.:	QC_0.6_1_	QC_0.6_2_	QC_0.6_3_	QC_1.2_1_	QC_1.2_2_	QC_1.2_3_
2	NA	RT	HY	HY	HY	HY	HY	Blank	SQC_0.6	SQC_0.6	SQC_0.6	SQC_1.2	SQC_1.2	SQC_1.2
3	0	10.435	84331	292570	67378	180646	206499	70613	145042	120732	119852	50767	118558	177556
4	1	11.928	46848	34305	34132	31465	45991	455	906	2961	554	622	420	1237
5	2	8.755	58875	30306	41297	25782	32535	159	1751	2426	1573	4100	3747	3526
6	3	11.667	46848	34305	34132	31465	6073	1395	8410	9735	11464	14186	12068	8153
7	4	21.371	655568	993115	717809	1046786	777284	1829	409794	393835	398060	557165	588185	537469
8	5	20.698	292966	268394	290298	259448	348828	6435	35932	27056	33865	45285	46479	43874
9	6	2.676	255063	241394	350466	227172	179186	85795	53309	25192	28126	77106	91666	77192
10	7	11.032	318056	266678	260501	255852	261064	135272	147772	101718	116589	130748	132229	159905
11	8	10.592	185072	24259	31019	34249	10783	9963	32492	62272	42541	103692	35754	77544
12	9	8.839	317054	352222	310879	372299	424599	68629	62133	411400	103548	423826	210173	459846
13	10	9.682	1938377	1807527	1643931	1140346	1432538	202989	235423	213754	246567	314386	259188	215493
14	11	11.336	35874	42521	49117	33813	29676	182591	102499	96297	96998	101850	86394	106002
15	12	4.652	240724	178612	268728	91454	279625	153621	218319	384326	355086	366204	412174	367197

Figure 2: Feature table format required for the following data preprocessing steps.

In particular, the second row should have the following labels so that the software can automatically recognize the sample type:

- **Blank** for method blanks
- **RT** for retention time
- **QC** for quality control samples
- **SQC_XX** for serial quality control samples
  - The XX represents the loading amount. For example, if 2 uL was used, then the appropriate label would be SQC_2.

5. Install prerequisite software

For the rest of this protocol, users will need to download and install the following software as data preprocessing will use R:

- R language https://www.r-project.org
- R studio https://posit.co

In this workflow, we will need to use the "MAFFIN", "nortest", "ggplot2", and "pracma" packages. To install and load these packages, the following code can be used:

```
#To install packages, remove the '#' symbol in the next four lines
#devtools::install_github("Waddlessss/MAFFIN")
#install.packages("nortest")
#install.packages("pracma")
#install.packages("ggplot2")

#Load libraries
library("MAFFIN")
library("nortest")
library("pracma")
library("ggplot2")
```

6. Load feature table into R

The working directory first needs to be changed to the folder that contains the input feature table. The file path should be specified as shown here:

```
#Set working directory. Remove the '#' symbol in
#the next line and specify your path following
#the format provided.
#setwd("C:/Users/User/Desktop/test")
```

Finally, the feature table will be loaded into R by using its file name. For example, if the name of the feature table is "ft.csv", then we will use the following code to save the feature table into the variable name `data`.

```
#Read in feature table
data = read.csv("ft.csv")
```

7. Filter feature table

Prior to sample normalization, two important data filtering procedures needs to be conducted to improve quantitative performance. They include 1) removing non-biological features; 2) removing metabolic features with poor quantitative performance. These two steps will exclude metabolic features that are unlikely to be meaningful about sample amounts, improving the performance of sample normalization.

First, non-biological features refer to the features coming from sample preparation and LC-MS analysis. We can remove them by comparing the detected features to the method blank because biologically meaningful features should be found at much higher levels in the biological samples. In addition, we can exclude features not within the analytical gradient. Second, metabolic features with poor quantitative performance refers to features with poor reproducibility in the QC samples and poor linearity in the serial QC samples. Features with a relative standard deviation (RSD) larger than 25% in the QC samples are excluded as they are not reproducible. In addition, features that are likely to be from noise or saturated signals should also be removed. We can recognize them using the serial QC samples. Thus, features with low correlation between injection volumes and MS signal intensities will also be removed.

Conveniently, users can exclude both non-biological features and features with poor quantitative performance with the *FeatureSelection()* function in the MAFFIN R package.

```
data = FeatureSelection(data)
```

8. Intensity corection

Because of fold-change bias in ESI response, MS intensity ratios do not always accurately reflect concentration ratios. Further details about fold-change bias can be found in our previous papers: https://pubs.acs.org/doi/10.1021/acs.analchem.0c00246 and https://pubs.acs.org/doi/abs/10.1021/acs.analchem.0c04113. To address this concern, MS signal intensities can be converted to loading amounts by constructing calibration curves with the serial QC samples. This step can be easily achieved using the *IntCorrection()* function from MAFFIN.

```
data = IntCorrection(data)
```

Afterwards, users can obtain an intensity-corrected feature table as shown below.

	Alignment.ID	RT	HY_1_P1.C.1_01_14367	HY_2_P1.C.2_01_14368	HY_3_P1.C.3_01_14369	HY_4_P1.C.4_01_14370	HY_5_P1.C.5_01_14371	MB_P1.B.2_01_14366	QC_0.6_1_P1.B.1_01_14391	QC_0.6_2_P1.B.1_01_
1	NA	RT	HY	HY	HY	HY	HY	Blank	SQC_0.6	SQC_0.6
2	0	10.435	84331	292570	67378	180646	206499	70613	145042	120732
3	1	11.928	46848	34305	34132	31465	45991	455	906	2961
4	2	8.755	58875	30306	41297	25782	32535	159	1751	2426
5	3	11.667	46848	34305	34132	31465	6073	1395	8410	9735
6	4	21.371	1.96115814428426	5.51609936351349	2.40493165041695	6.70398563407198	2.88247899634398	1829	409794	393835
7	5	20.698	10.6503153796318	9.52592003372319	10.5254291116605	9.13006083625007	13.4535486185095	6435	35932	27056
8	6	2.676	5.51198513949857	5.12812959668487	8.19111092419794	4.72874459925298	3.38119232202287	85795	53309	25192
9	7	11.032	318056	266678	260501	255852	261064	135272	147772	101718
10	8	10.592	9.6195497232981	0.206746279395147	0.319053086212857	0.374505894706056	0.141359746549652	9963	32492	62272
11	9	8.839	317054	352222	310879	372299	424599	68629	62133	411400
12	10	9.682	1938377	1807527	1643931	1140346	1432538	202989	235423	213754
13	11	11.336	35874	42521	49117	33813	29676	182591	102499	96297
14	12	4.652	240724	178612	268728	91454	279625	153621	218319	384326
15	13	1.962	46771	50134	52427	31389	65612	146378	281391	64101
16	14	8.684	331421	336140	308363	332348	367913	65410	328451	320999
17	15	2.281	2.42085290322673	1.37627976760477	0.968392995224623	1.70963016488667	2.17390280233897	119033	281391	235372

Figure 3: Intensity corrected feature table.

9. Filter missing values

It is common for some metabolic features to have a high missing rate across samples. These features are often less reliable and may represent noise. Thus, we can use the following code to remove features with

too many missing values. By default, we will remove features with more than 50% missing values. The percentage can be adjusted by changing the `filter_missing` parameter.

```
#Features with >filter_missing percentage missing values will be removed
filter_missing = 0.5

#Clean up feature table and remove features with too many missing values
norm_df = data[grep("high|FeatureQuality", data$Quality),]
norm_df = norm_df[,-grep("RT|SQC|QC|Blank|FeatureQuality|Model", norm_df[1,])]
norm_df = norm_df[c(1,which(rowSums(norm_df == 0) <= round((ncol(norm_df) - 1)*filter_missing))),]
```

Now, the feature table `norm_df` is ready for sample normalization.

10. Evaluate symmetry factors and choose the normalization algorithm

Unbalanced data is a phenomenon where the percentage of up- and downregulated metabolic features between two samples are unequal. This data structure greatly impacts the performance of normalization methods. Thus, the severity of unbalanced data should be considered when deciding on which normalization method to use. To guide users on which normalization method to use, we proposed a metric called the symmetry factor (SF). This metric is used to evaluate how unbalanced each sample is with respect to a reference sample. The reference sample is defined as the sample with the least number of missing values. In the case of unbalanced data, maximal density fold change (MDFC) is recommended for sample normalization. In the case of balanced data, both probabilistic quotient normalization (PQN) and MDFC are good options.

Here, we provide the code to find the reference sample and the function to evaluate SFs:

```
#Find reference sample
ref_index = names(sort(colSums(norm_df == 0), decreasing = F)[1])

#Function to find SFs
find_unbalanced = function(data){

  group_vector = as.character(data[1,])[-1]
  working_df = sapply(data[-1,-1], as.numeric)
  working_ref = which(colnames(working_df) == ref_index)
  sf = c()

  for(i in 1:ncol(working_df)){

    if(i == working_ref){
      next
    }

    fc = working_df[,i]/working_df[,working_ref]

    #Take only values detected in both samples

    fc = fc[is.finite(log(fc))]

    #Do not log transform if already normal

    if(ad.test(fc)$p.value < 0.05){
```

```
      #Log transform

      fc = log2(fc)
      fc = fc[is.finite(fc)]

    }

    #Kernel density estimation

    d = density(fc, bw = 0.3, n = 2^15)

    #Find FC with maximal density and integrate
    #to the right and left of it

    peak_index = which.max(d$y)
    area_left = trapz(d$x[1:peak_index],d$y[1:peak_index])
    area_right = trapz(d$x[peak_index:length(d$x)],d$y[peak_index:length(d$x)])

    #Symmetry factor

    sf[i] = area_right/area_left
  }

  #Condition where the last sample was the reference

  if(working_ref == ncol(working_df)){
    sf = c(sf, NA)
  }

  #Store results

  results = data.frame(colnames(working_df),group_vector, sf)
  colnames(results) = c("id","grouping", "sf")

  #Remove reference sample

  results = results[-working_ref,]

  return(results)
}
```

We will now use the *find_unbalanced* function to calculate SFs on our feature table `norm_df`. Let's also take a look at the first few rows of the results.

```
screen_unbalanced = find_unbalanced(norm_df)
head(screen_unbalanced)
```

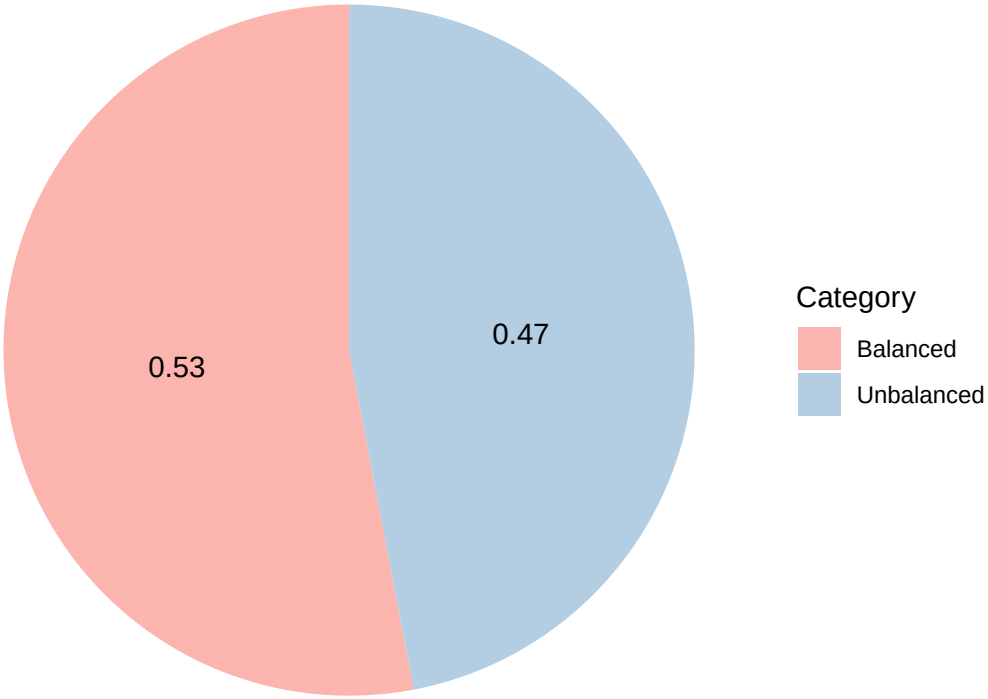
```
##                     id grouping        sf
## 1 HY_1_P1.C.1_01_14367       HY 0.8762947
## 2 HY_2_P1.C.2_01_14368       HY 1.0252580
## 3 HY_3_P1.C.3_01_14369       HY 1.0161818
## 4 HY_4_P1.C.4_01_14370       HY 1.0702970
## 5 HY_5_P1.C.5_01_14371       HY 0.7834032
```

```
## 6 SW_1_P1.D.2_01_14378       SW 0.8048669
```

We can see that for each sample, a SF was calculated. Let's use a pie chart to better visualize what percentage of samples in our dataset have unbalanced data.

```
#Get the percentage of samples with unbalanced data
perc_balanced = round(mean(screen_unbalanced$sf >=
                        0.8 & screen_unbalanced$sf <= 1.25),2)
perc_unbalanced = 1 - perc_balanced

#Pie chart
pie_df = data.frame(label = c("Balanced", "Unbalanced"),
                    percent = c(perc_balanced, perc_unbalanced))
ggplot(pie_df, aes(x = "", y = percent, fill = label)) +
  geom_bar(width = 1, stat = "identity") +
  coord_polar("y") +
  theme_void() +
  labels(fill = "Category") + scale_fill_brewer(palette = "Pastel1") +
  geom_text(aes(label = percent), position = position_stack(vjust = 0.5))
```



Using SFs, we found that 47% of our samples have unbalanced data. Unless the dataset has a low percentage of samples with unbalanced data (i.e., less than 10%), then MDFC should be used. If a suboptimal normalization method is used, then statistically significant features between experimental groups could never be detected. In addition, suboptimal normalization methods can introduce false positives, potentially resulting in misleading biological interpretations. Thus, in this example, we will use MDFC. Notably, the reference sample used for calculating SFs will also be used during normalization. Thus, the intensities of all samples

will be adjusted to this reference sample. In the final output, metabolic features that were initially filtered out have been included back, allowing users to further explore them if desired.

The following are the functions for PQN and MDFC normalization.

```
#Functions to normalize
#Note the input of the function requires two inputs:
#data contains all the features while norm_df only
#contains the high-quality features that will be used for
#calculating normalization factors
PQN_norm = function(data, norm_df){

  #Quick check to make sure the right data is being used

  if(nrow(norm_df) > nrow(data)){
    stop("Make sure that the input of data and norm_df are correct.")
  }

  #Index reference sample

  norm_df = sapply(norm_df[-1,-1], as.numeric)
  ref = which(colnames(norm_df) == ref_index)

  #Calculate normalization factors

  estimated_k = c()

  for(i in 1:ncol(norm_df)){
    fc = norm_df[,i]/norm_df[,ref]
    fc = fc[is.finite(log2(fc))]
    estimated_k[i] = median(fc)
  }

  #Find the corresponding sample in the original feature table and normalize

  for(i in 1:length(estimated_k)){
    sample_index = which(colnames(norm_df)[i] == colnames(data))
    normalized_int = as.numeric(data[2:nrow(data),sample_index])/estimated_k[i]
    data[2:nrow(data), sample_index] = normalized_int
  }

  #Return normalized data

  return(data)

}
MDFC_norm = function(data, norm_df, bw = 0.3){

  #Check to make sure the right data is being used

  if(nrow(norm_df) > nrow(data)){
    warning("Make sure that the input of data and norm_df are correct.")
  }

  #Index reference sample
```

```
  norm_df = sapply(norm_df[-1,-1], as.numeric)
  ref = which(colnames(norm_df) == ref_index)

  #Calculate normalization factors

  estimated_k = c()

  for(i in (1:ncol(norm_df))[-ref]){
    fc = log2(norm_df[,i]/norm_df[,ref])
    fc = fc[is.finite(fc)]
    d = density(fc, bw = bw)
    estimated_k[i] = 2^d$x[which.max(d$y)]
  }

  estimated_k[ref] = 1

  #Find the corresponding sample in the original feature table and normalize

  for(i in 1:length(estimated_k)){
    sample_index = which(colnames(norm_df)[i] == colnames(data))
    normalized_int = as.numeric(data[2:nrow(data),sample_index])/estimated_k[i]
    data[2:nrow(data), sample_index] = normalized_int
  }

  #Return normalized data

  return(data)

}
```

As discussed above, MDFC will be used as the SFs indicated that a large percentage of samples have unbalanced data:

```
normalized_df = MDFC_norm(data, norm_df)
```

11. Save normalized feature table

The normalized feature table `normalized_df` can be saved in the working directory using the following code:

```
#Save feature table
write.csv(normalized_df, "norm_ft.csv", row.names = F)
```

The normalized feature table (as shown below) is now ready for downstream quantitative analysis.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Alignmen	RT	HY_1_P1.(	HY_2_P1.(	HY_3_P1.(	HY_4_P1.(	HY_5_P1.(	MB_P1.B.2	QC_0.6_1_	QC_0.6_2_	QC_0.6_3_	QC_1.2_1_	QC_1.2_2_	QC_1.2_3_
2	NA	RT	HY	HY	HY	HY	HY	Blank	SQC_0.6	SQC_0.6	SQC_0.6	SQC_1.2	SQC_1.2	SQC_1.2
3	0	10.435	77030.48	430344.7	66476.6	235609.4	194021.6	70613	145042	120732	119852	50767	118558	177556
4	1	11.928	42792.38	50459.64	33675.37	41038.55	43212.07	455	906	2961	554	622	420	1237
5	2	8.755	53778.2	44577.46	40744.52	33626.44	30569.12	159	1751	2426	1573	4100	3747	3526
6	3	11.667	42792.38	50459.64	33675.37	41038.55	5706.048	1395	8410	9735	11464	14186	12068	8153
7	4	21.371	1.791381	8.113697	2.372758	8.743741	2.70831	1829	409794	393835	398060	557165	588185	537469
8	5	20.698	9.72832	14.01179	10.38462	11.90797	12.64064	6435	35932	27056	33865	45285	46479	43874
9	6	2.676	5.034814	7.543028	8.081528	6.167513	3.176889	85795	53309	25192	28126	77106	91666	77192
10	7	11.032	290522	392259.9	257016	333697.6	245289.6	135272	147772	101718	116589	130748	132229	159905
11	8	10.592	8.786787	0.304106	0.314785	0.488453	0.132818	9963	32492	62272	42541	103692	35754	77544
12	9	8.839	289606.7	518087.6	306720	485574.8	398943.3	68629	62133	411400	103548	423826	210173	459846
13	10	9.682	1770572	2658713	1621938	1487308	1345979	202989	235423	213754	246567	314386	259188	215493
14	11	11.336	32768.4	62544.65	48459.9	44100.95	27882.87	182591	102499	96297	96998	101850	86394	106002
15	12	4.652	219884.6	262722.5	265132.9	119279.8	262729.1	153621	218319	384326	355086	366204	412174	367197

Figure 4: Normalized feature table.