An Introduction to the rawDiag R Package

Christian Trachsel FGCZ
Tobias Kockmann FGCZ
Christian Panse FGCZ

Abstract

rawDiag is an R package which provides the user with visualizations of mass spectrometry data characteristics, coming from LC-MS/MS proteomics experiments. The plots implemented in this package allow the user to check LC-MS/MS method parameters and facilitates the optimization thereof. The package is developed, tested and used at the Functional Genomics Center Zurich. The package is optimized for reading Thermo Fisher Scientific raw files, but any other mass spectrometry vendor data format is supported over the open standards using the **mzR** package. **rawDiag** can be run on modern laptop infrastructure with sufficient speed, but for large datasets, a server infrastructure is beneficial.

Keywords: proteomics, mass spectrometry, visualization, method optimization, R-package.

1. Introduction

Mass spectrometry is a well-accepted and widespread method in life sciences. An important task that needs to be performed prior to acquisition of any data set is the optimization of the applied mass spectrometry method. **rawDiag** (Trachsel, Panse, Kockmann, Wolski, Grossmann, and Schlapbach 2018) builds on the idea of the discontinued software rawMeat (Vast Scientific). Our software allows a mass spectrometrist to analyze raw files in a short amount of time and produces diagnostic plots of LC-MS/MS run chararcteristics as result. These visualizations are helpful for the optimization of the instrument method towards the sample at hand.

Our R package can read directly from the instrument raw data and keeps the data in a `data.frame` object where each row represents a scan event and each column contains LC-MS/MS run parameters. The resulting R `data.frame` is structured as tidy data based on Wickham (2014). **rawDiag** is optimized to work with Thermo Fisher Scientific raw files. The package calls a C# executable that generates the required `data.frame` on the fly directly from a raw file using the New RawFileReader .Net assembly (Shofstahl 2018). However, any other mass spectrometry data format is also supported via the open standards using the R package **mzR** (Fischer and Neumann 2017). This requires a conversion of the data to e.g. mzML prior to loading into the R session with the addapter function `as.rawDiag.mzR`. Examples for reading data is explained in section 3 of this vignette. A number of R helper functions reshape and subset the data and pass the desired data-frames to **ggplot2** (Wickham 2009) for visualization. Different kinds of visualizations (Trellis-like, violin plots, overlay plots) help to compare result between different runs. Mass spectrometry data can be inspected interactively running the software as an R shiny instance. Additionally, pdf reports can be generated using

a customizable **rmarkdown** file.

Processing speed of the software on a modern laptop infrastructure is sufficient to provide the user with results already a few minutes after the mass spectrometry data was acquired (a single file containing ≈ 80′000 individual spectra is processed in less than 50 sec on a 2018 MacBook). A large scale performance benchmark can be found in section 5.

All necessary steps can be done with the R command line or by using the shiny application.

The diagnostic visualization are peptide ID free and rely on data logged by the instrument directly in the raw data file. This makes the software slim and fast and does not require additional data analysis pipeline.

2. Getting started

help can be found by checking the documentation.

```
R> help(package="rawDiag")
```

attach the package

```
R> library(rawDiag)
```

3. Getting the Data

3.1. Utility Functions

We first load the demonstration data shipped with the package. Please note, the raw data are available through MassIVE MSV000082389.

load sample data:

```
R> data(WU163763)
R> stopifnot(is.rawDiag(WU163763))

R> data("WU163763")
R> is.rawDiag(WU163763)

[1] TRUE

R> names(WU163763)

 [1] "filename"           "scanNumber"         "StartTime"
 [4] "BasePeakMass"       "BasePeakIntensity"  "TIC"
 [7] "ScanType"           "CycleNumber"        "MSOrder"
[10] "MassAnalyzer"       "PrecursorMass"      "AGC"
[13] "ChargeState"        "IonInjectionTimems" "FTResolution"
[16] "LMCorrection"       "PrescanMode"        "MasterScanNumber"
[19] "ElapsedScanTimesec" "AGCMode"            "transient"
```

3.2. Using The New RawFileReader

The function `read.raw` uses the .Net assembly (Shofstahl 2018) based executable to directly extract the information out of the mass spectrometry meassurement file. The C# source code as well as the compiler and linker options for a Linux build is provided as a docker recipe (see `inst/docker/Dockerfile`. We use R for the implementation and read the data using the pipe command

The following code snippet demonstrates the use of the `read.raw` function to extract data from a tiny example raw file:

```
R> rawfile <- file.path(path.package(package = "rawDiag"),
+                        "extdata", 'sample.raw')
R> system.time(RAW <- read.raw(file = rawfile))

   user  system elapsed
  0.472   0.063   0.453

R> summary.rawDiag(RAW)

             Ms Ms2
  sample.raw  27 546
```

reading all parameters

```
R> dim(RAW)

[1] 573  21

R> RAW <- read.raw(file = rawfile, rawDiag = FALSE)
R> dim(RAW)

[1] 573  82
```

Additional information can be found using the R man pages.

```
R> ?read.raw
```

3.3. Reading the Open Proteomics Standard Files

The package also ships with an adapter function `rawDiag:::as.rawDiag.mzR` which enables the support of open file standards by using the Bioconductor **mzR** package (Fischer and Neumann 2017).

The following example code shows the access on a mzML file.

```
R> library(mzR);
R> mzML <- "04_S174020_5000_5010.mzML"
R> mzML <- file.path(path.package(package = "rawDiag"), "extdata", mzML)
R> system.time(RAW <- rawDiag:::as.rawDiag.mzR(openMSfile(mzML)))
```

```
   user  system elapsed
  0.100   0.002   0.103
```

```
R> summary.rawDiag(RAW)
```

```
                             Ms Ms2
  04_S174020_5000_5010.mzML  3   8
```

```
R> RAW$scanNumber
```

```
 [1] 5000 5001 5002 5003 5004 5005 5006 5007 5008 5009 5010
```

4. Usage

In the following section we demonstrate how the software can be used for the optimization of one parameter in an LC-MS/MS method. For this we load the data included in the package:

```
R> data(WU163763)
R> stopifnot(is.rawDiag(WU163763))
```

This was recorded to investigate the optimal number of MS2 scans between two consecutive MS1 scans on a Q-Exactive HF-X mass spectrometer. A commercially available HeLa digest was measured with a generic starting method and two modified methods. Each method was analyzed three times in randomized order. For demonstration purposes, we first filter the data for a single injection per method, which reduces potential overplotting issues at the same time. The filtering is done with the following code snippet:

```
R>   library(tidyverse)
R> df <- dplyr::filter(WU163763,
+    filename == "04_S174020" |
+    filename == "05_S174020" |
+    filename == "09_S174020")
```

Our generic starting method is performing 18 MS2 scans for each MS1 scan. With the known time needed for a single scan on the used instrument we can calculate the cycle time (time required to scan one MS1 and 18 MS2 scans). In our case we find the cycle time to be ≈ 0.6 second. For a 120 min chromatography with expected peak width of 20-30 seconds, this would result in 30-50 MS1 points per peak. In other words the instrument would spend too much time recording MS1 compared to MS2. The hypothesis for an optimized method is that we need to give the instrument the opportunity to performe more MS2 scan between two MS1 scans. Therefore we addapted the methods to perform 36 and 72 MS2 scans respectively. Inorder to check the effect of these modifications, data needs to be recorded with the three methods. After the data is recorded we can visualize and analyse the results with the dignostic plots of this package.

```
R> print(gp <- PlotTicBasepeak(df, method = "overlay"))
```

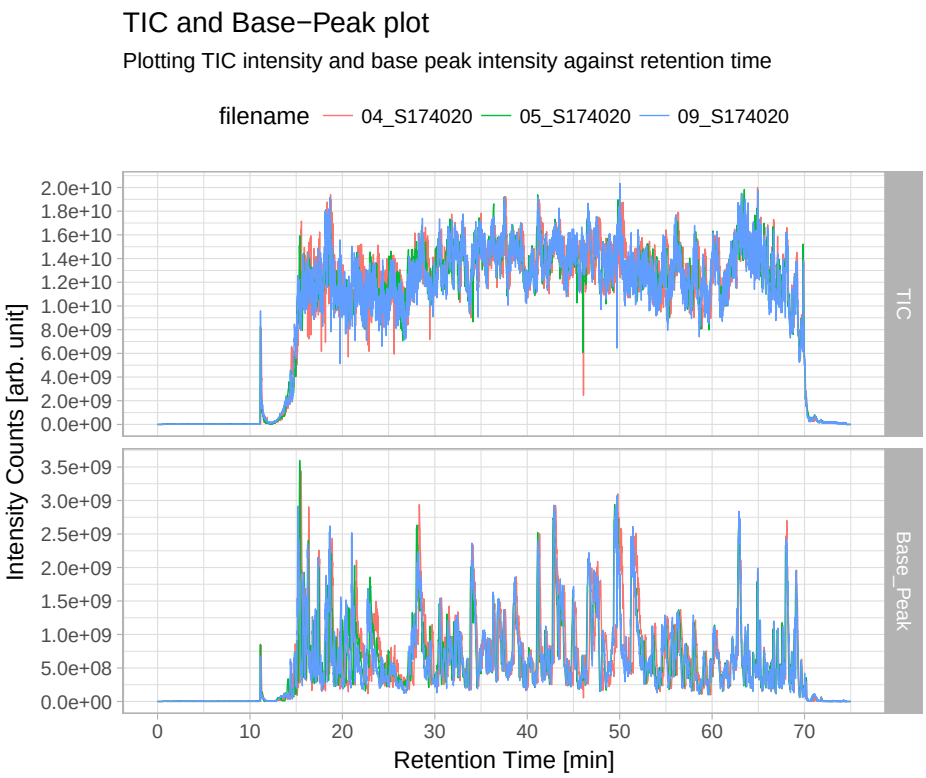

Figure 1: TIC and Base Peak Plot – This plot shows the overlay of the total ion chromatogram (TIC) and the base peak chromatogram of the three used mass spectrometry runs. In all three injections, the TIC and base peak response is close to identical which indicated that both the liquid chromatography as well as the mass spectrometry signal response for the three measurements was similar.

```
R> print(gp <- PlotCycleTime(df, method = "trellis") +
+    facet_wrap(~ filename, ncol =1))
```

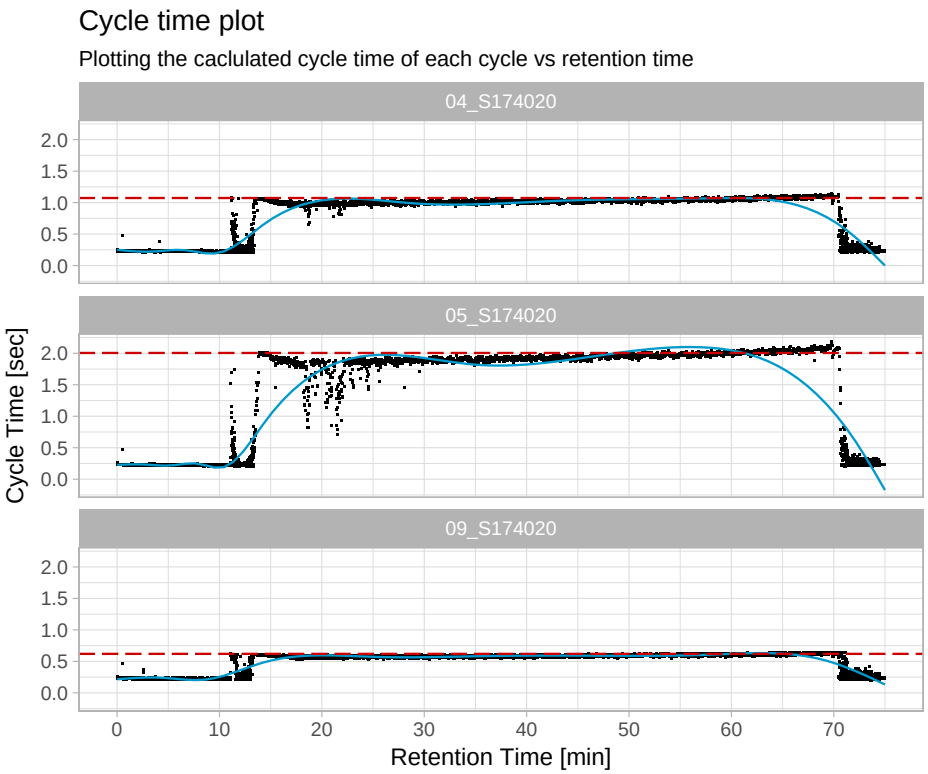

Figure 2: Cycle Time Plot – This plot displays the empirical cycle time for each recorded duty cycle in a method as a dot plot. The blue line represents a fitted trend using a gam model. The red dashed line is indicating the 95th quantile.

The first parameter we want to check is the TIC or base peak. With this plot we can see if the data was recorded properly and if the signal response of the sample is the same over the three injections.

As shown in Figure 1, the signal response and chromatography is ok for the three injections as they are almost identical. This indicates a stable chromatography and an similar signal response of the three measurements. This behaviour is the pre-requisit for comparing the three runs with each other. In case of large deviations, affected runs would need to be excluded from further analysis or the data set should be re-analyzed. As next point, we want to check the actual cycle time of the three measurements. Please note, here we also make use of the grammar of graphics (Wickham 2009) implementation of **ggplot2** by modifiying the plot appearance on the fly. We move grey box indicating the faceting labels from the standard position on the right side of to the top. This is achieved by adding the additional `facet_wrap` statement after the initial call of the plot function.

In Figure 2 we can see the the cycle time for the top 18 method (file: 09_S174020) is arround 0.6 seconds during the elution phase of the peptides. This is in good agreemnet with our theoretically calculated value. The red dashed line in this plot indicates the 95th quantile.

```
R> print(gp <- PlotCycleLoad(df, method = "trellis"))
```

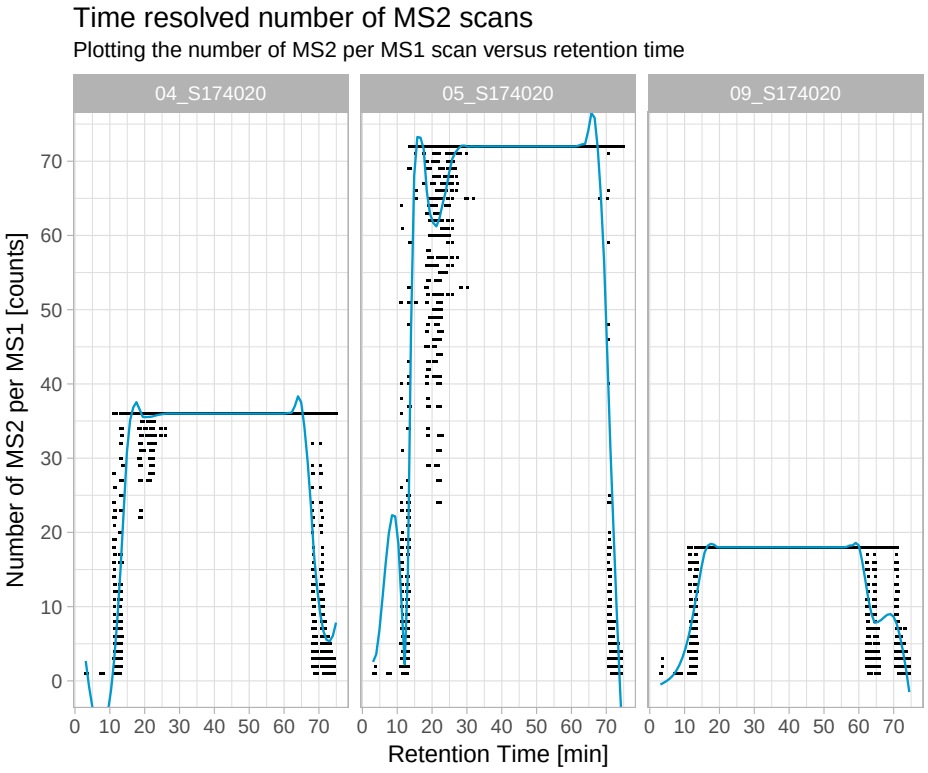

Figure 3: Cycle load plot – The plot shows number of MS2 scans performed after each MS1 scan. The value should be at or close to the maximum during the main peptide elution phase. In all three mehtods this is the case. If the value would be lower, this would indicate that the method is not well addapted towards a) the sample complexity or b) the amount of sample injected.

As initially mentioned, the generic starting method is producing too many MS1 data points under the used chromatography settings. This is verified by the empirical cycle time. The cycle time of the top 36 (file: 04_S174020) and top 72 (file: 05_S174020) methods respectively have cycle times of 1.1 second and 2 seconds respectively. With the used chromatography we produce ≈20-30 MS1 point for the top 36 and ≈10-15 MS1 points for the top 72 method per chromatographic feature. The value from the top 72 mehtod is is a better value compared to the value of the initial method. 10-15 MS1 data points are considered as a good balance between time spent on MS1 and MS2 scans and allow a deep sampling of the peptides in the sample. The next thing to check is if the instrument is actually using the available MS2 capacity. For this, we plot the cycle load (the actually performed MS2 scans for each MS1 scan).

From Figure 3, we can see that all methods actually use the maximum available number of MS2 scans during the main peptide elution phase. The blue line is representing the trend as a fitted gam model. The fact that the top 72 method uses all the available MS2 slots, indicates that the two other methods are not optimally analyzing the investigated sample. The complexity is too high for these two methods and after selecting the precursors for

```
R> print(gp <- PlotInjectionTime(subset(df, MSOrder %in% "Ms2"),
+      method = "violin")
+    )
```

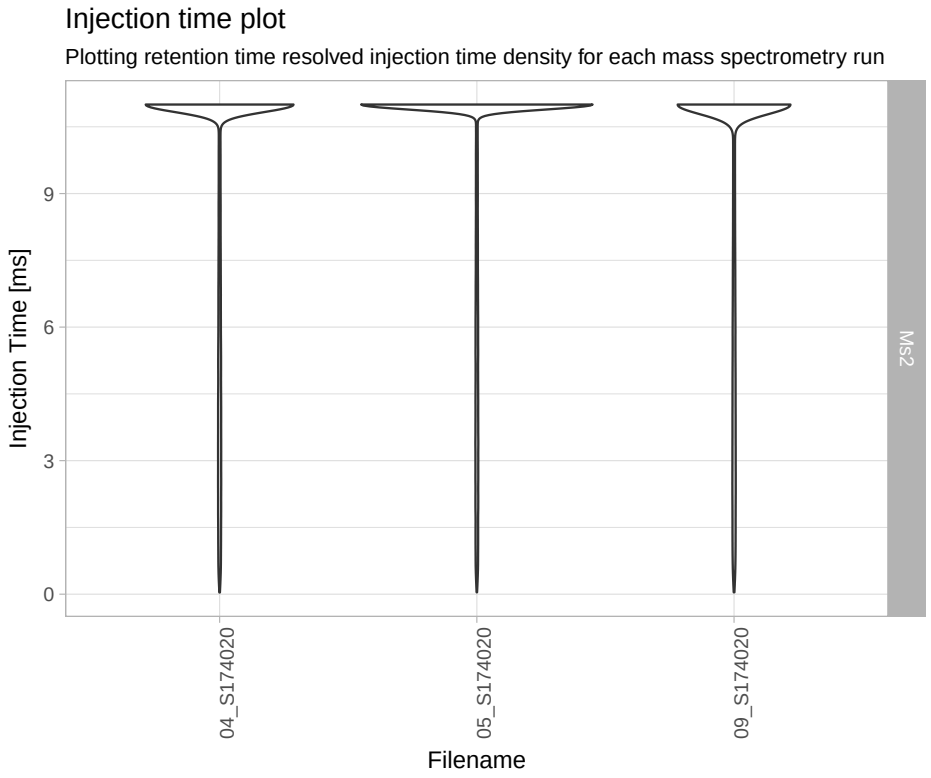

Figure 4: Injection Time Plot – The plot shows the injection time density of each mass spectrometry file as a violin plot. The higher the maximum number of MS2 scans is in the method, the more the density is shifted towards the maximum injection time value.

MS2 fragmentationt at any give time during the peptide elution, there are still many other potential candidates available which are not fragmented. However, we can not endless increase the amount of MS2 scans, since we want to maintain a certain cycle time. In order to have more than 72 MS2 scans we would need to switch to shorter individual scans. But this would limit the amount of time the instrument has to collect the ions needed for each individual scan. A to low amount for this parameter (injection time on Orbitrap instruments) would lead to bad fragmentation spectra quality and as a direct result to a lower amount of identified peptides. To check if the three used mehtods are still operating in an acceptable injection time regime, we use the `PlotInjectionTime` function.

From Figure 4 one can see that the injection time distribution is shiftet towards longer injection times, the more MS2 scans are available. This behaviour is kind of expected, since the instrument is picking more lower abundat ions to fill all the available MS2 slots in the top 36 and top 72 methods. From the plot itself the distribution indicated that the top 72 method is still operating in a acceptable range.

At this point, we can now perform a database search with the three files to check if we identify

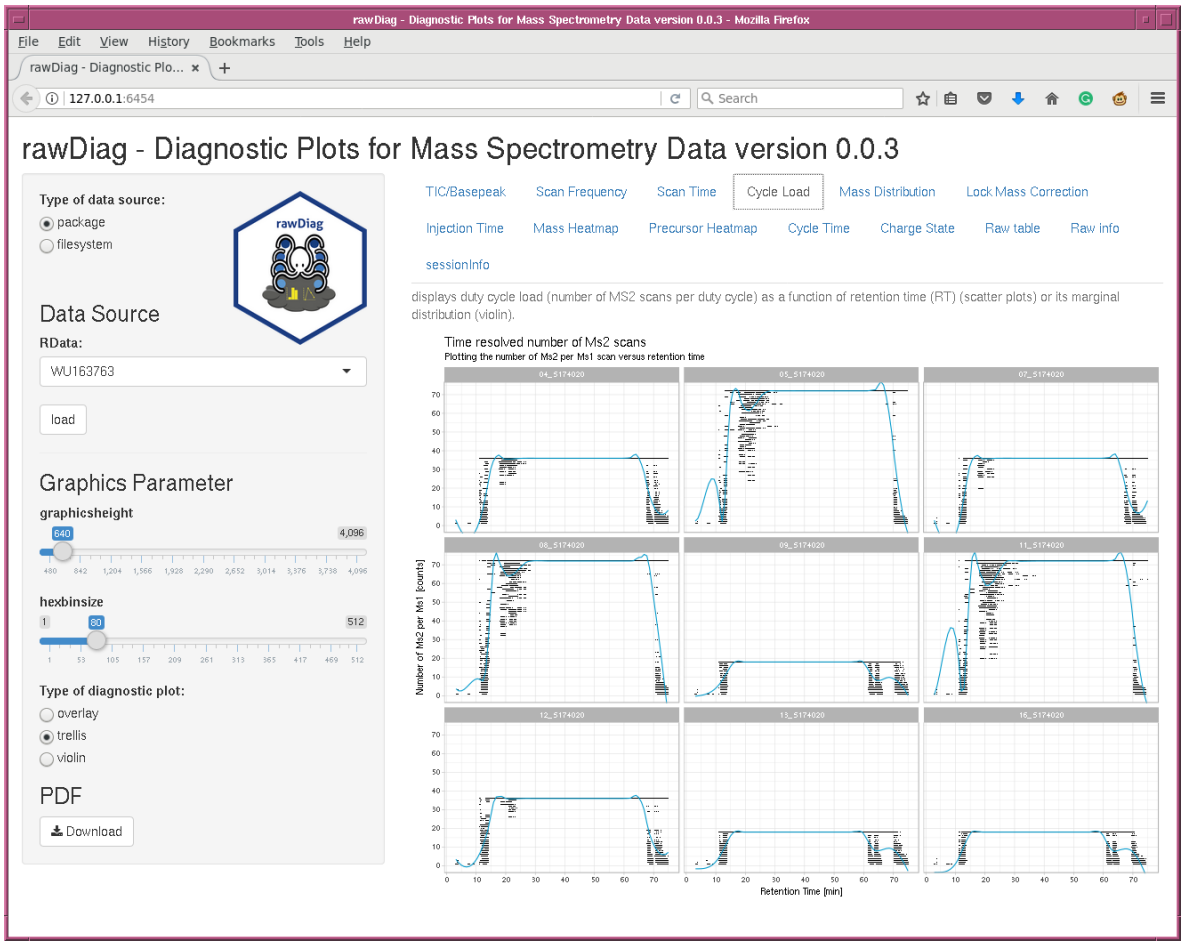

Figure 5: Running the **rawDiag** shiny application.

more peptide and proteins with the two optimized methods. We can also use this additional data to check if the quality of the spectrs is starting to decrease and if further optimizations are required on the mehtods. All the described steps executed on the command line in an R session can also be performed in an interactive Shiny environment. To start the Shiny application (see Figure 5) type the following into your R shell:

```
R> # install.packages("shiny"); install.packages("DT")
R> library(shiny)
R> rawDiag_shiny <- system.file('shiny', 'demo', package = 'rawDiag')
R> shiny::runApp(rawDiag_shiny, display.mode = 'normal')
```

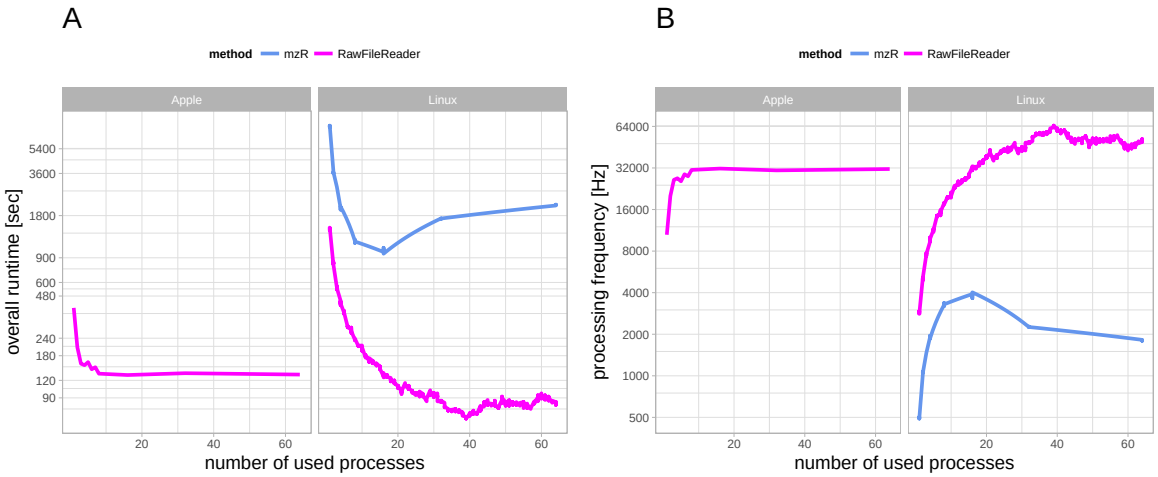

Figure 6: Benchmark – The left plot shows the overall logarithmic scaled runtime of 128 raw files. The graphic on the right side shows the derived IO throughput as scan information per second. The plots illustrate that both systems, server, and laptop, can analyze 95GB of instrument data within less than three minutes.

5. Benchmark

As described in Trachsel *et al.* (2018) we performed a benchmark on different plattforms using a larger data set by the following code snippet.

```
R> library(parallel)
R> f <- list.files()
R> f <- f[grep("raw$", f)]
R> b <- lapply(1, function(x){rawDiag:::benchmark_raw(f,
+    exe="~/RiderProjects/fgcz-raw/bin/Debug/fgcz_raw.exe")})
R> b <- plyr::rbind.fill(lapply(b, plyr::rbind.fill))
R> b$overall.runtime <- as.integer(format(b$end.time, "%s")) -
+    as.integer(format(b$start.time, "%s"))
R> b$system <- "Linux"
R> b.Linux <- b
R> save(b.Linux, file='benchmark.RData')
```

We performed the benchmark using the proteomeXchange PXD006932 data set. The performance metrics are visulized in Figures 6.

6. Developer notes

6.1. Build R package

To build the R package including the vignettes files requires the Thermo dll-files (Shofstahl 2018) have to be placed next to the executable `fgcz_raw.exe` in the package's `exec` directory prior to the build process.

A code snippet on how to register the .Net assemblies on a Linux system is shown below:

Listing 1: code snippet register .Net assemblies.

```
cd /tmp/ \
  && unzip /tmp/ThermoRawFileReader_linux.4.0.22.nupkg \
  && gacutil -i lib/ThermoFisher.CommonCore.BackgroundSubtraction.dll \
  && gacutil -i lib/ThermoFisher.CommonCore.Data.dll \
  && gacutil -i lib/ThermoFisher.CommonCore.RawFileReader.dll
```

An example of how to compile and link the C# code for the `read.raw` wrapper function accessing Thermo instrument files on a Linux platform using the mono enviroment is shown below.

Listing 2: compile and link C# code on Linux.

```
mcs /out:fgcz_raw.exe \
  fgcz_raw.cs /r:lib/ThermoFisher.CommonCore.Data.dll \
  /r:lib/ThermoFisher.CommonCore.MassPrecisionEstimator.dll \
  /r:lib/ThermoFisher.CommonCore.RawFileReader.dll /target:exe
```

6.2. Known issues

- the used `system2` function on windows R version 3.5.0 is not writting into a file.

7. Session information

An overview of the package versions used to produce this document are shown below.

- R version 3.4.2 (2017-09-28), `x86_64-apple-darwin15.6.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Running under: `macOS High Sierra 10.13.6`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: bindrcpp 0.2.2, dplyr 0.7.5, forcats 0.3.0, ggplot2 2.2.1, mzR 2.10.0, purrr 0.2.5, rawDiag 0.0.10, Rcpp 0.12.16, readr 1.1.1, stringr 1.3.0, tibble 1.4.2, tidyr 0.8.0, tidyverse 1.2.1
- Loaded via a namespace (and not attached): assertthat 0.2.0, bindr 0.1.1, Biobase 2.36.2, BiocGenerics 0.22.1, bit 1.1-14, bit64 0.9-7, blob 1.1.1, broom 0.4.4, cellranger 1.1.0, cli 1.0.0, codetools 0.2-15, colorspace 1.3-2, compiler 3.4.2,

crayon 1.3.4, DBI 0.8, digest 0.6.15, foreign 0.8-69, glue 1.2.0, grid 3.4.2, gtable 0.2.0, haven 1.1.1, hexbin 1.27.2, hms 0.4.2, httr 1.3.1, jsonlite 1.5, labeling 0.3, lattice 0.20-35, lazyeval 0.2.1, lubridate 1.7.4, magrittr 1.5, Matrix 1.2-11, memoise 1.1.0, mgcv 1.8-20, mnormt 1.5-5, modelr 0.1.1, munsell 0.4.3, nlme 3.1-131, parallel 3.4.2, pillar 1.2.1, pkgconfig 2.0.1, plyr 1.8.4, ProtGenerics 1.8.0, psych 1.8.3.3, R6 2.2.2, readxl 1.1.0, reshape2 1.4.3, rlang 0.2.0, RSQLite 2.1.0, rstudioapi 0.7, rvest 0.3.2, scales 0.5.0, stringi 1.1.7, tidyselect 0.2.4, tools 3.4.2, xml2 1.2.0

References

Fischer B, Neumann S (2017). "mzR." doi:10.18129/b9.bioc.mzr. URL https://doi.org/10.18129/b9.bioc.mzr.

Shofstahl J (2018). "New RawFileReader from Thermo Fisher Scientific." Version 4.0.22, URL http://planetorbitrap.com/rawfilereader#.WvWESK3QPmE.

Trachsel C, Panse C, Kockmann T, Wolski WE, Grossmann J, Schlapbach R (2018). "rawDiag - an R package supporting rational LC-MS method optimization for bottom-up proteomics." *bioRxiv*. doi:10.1101/304485. https://www.biorxiv.org/content/early/2018/04/24/304485.full.pdf, URL https://www.biorxiv.org/content/early/2018/04/24/304485.

Wickham H (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-0-387-98140-6. doi:10.1007/978-0-387-98141-3. URL http://ggplot2.org.

Wickham H (2014). "Tidy Data." *Journal of Statistical Software*, **59**(10). doi:10.18637/jss.v059.i10. URL https://doi.org/10.18637/jss.v059.i10.

Affiliation:

Christian Trachsel, Tobias Kockmann, Christian Panse
Functional Genomics Center Zurich
Swiss Federal Institute of Technology in Zurich | University of Zurich
Winterthurerstr. 190, CH-8057 Zurich, Switzerland

Telephone: +41-44-63-53912
E-mail: cp@fgcz.ethz.ch
URL: http://www.fgcz.ch