MSnbase, efficient and elegant R-based processing and visualisation of raw mass spectrometry data

Laurent Gatto,[*,†] Sebastian Gibb,[‡] and Johannes Rainer[¶]

†*Computational Biology Unit, de Duve Institute, Université catholique de Louvain, Brussels, Belgium*

‡*Department of Anaesthesiology and Intensive Care of the University Medicine Greifswald, Germany*

¶*Institute for Biomedicine, Eurac Research, Affiliated Institute of the University of Lübeck, Bolzano, Italy*

E-mail: laurent.gatto@uclouvain.be

Abstract

We present version 2 of the `MSnbase` R/Bioconductor package. `MSnbase` provides infrastructure for the manipulation, processing and visualisation of mass spectrometry data. We focus on the new *on-disk* infrastructure, that allows the handling of large raw mass spectrometry experiments on commodity hardware and illustrate how the package is used for elegant data processing, method development, and visualisation.

Keywords: R, Bioconductor, mass spectrometry, software, metabolomics, proteomics, visualisation, reproducible research

Introduction

Mass spectrometry is a powerful technology to assay chemical and biological samples. It is used in routine applications with well characterised protocols such as in clinical settings, as well as a development platform, with the aim to improve on existing protocols and devise new ones. The complexity and diversity of mass spectrometry yield complex data of considerable size, that require non trivial processing before producing interpretable results. The complexity and size of these data constitute a significant challenge for protocol development: in addition to the development of sample processing and mass spectrometry methods that yield the raw data, it is essential to process, analyse, interpret and assess these new data to demonstrate the improvement in the technical, analytical and computational workflows.

Practitioners have a diverse catalogue of software tools at their disposal. These range from low level software libraries that are aimed at programmers to enable the development of new applications, to more user-oriented applications with graphical user interfaces which provide a more limited set of functionalities to address a defined scope. Examples of software libraries include Java-based jmzML[1] or C/C++-based ProteoWizard.[2] Thermo Scientific Proteome Discoverer (Thermo Fisher Scientific), MaxQuant[3] and PeptideShaker[4] are among the most widely used user-centric applications.

In this software note, we present version 2 of the `MSnbase`[5] software, available from the Bioconductor[6] project. The package, like other software such as Python-based pyOpenMS,[7] spectrum_utils[8] or Pyteomics,[9] offers a platform that lies between low level libraries and end-user software. `MSnbase` provides a flexible R[10] command-line environment for metabolomics and proteomics mass spectrometry-based applications. It lays out a sound infrastructure to work with raw mass spectrometry data from MS files in mzML, mzXML, mzData or ANDI-MS/netCDF format as well as quantitative and proteomics identification data. The package enables manipulation (for example subsetting, filtering, or accessing specific parts thereof), detailed step-by-step processing (for example smoothing and centroiding of profile-mode MS data, or normalisation and imputation of quantitative data), analysis and visualisation

of these data and the development of novel computational mass spectrometry methods.[11] Extensive documentation and use cases are provided in *package vignettes*[12] and workflows.[13] Here, we focus on the new developments pertaining to raw mass spectrometry data handling and processing.

Infrastructure for raw data

In `MSnbase`, mass spectrometry experiments are handled as `MSnExp` objects. While the implementation is more complex, it is useful to schematise a raw data experiment as being composed of raw data, i.e. a collection of individual spectra, as well as spectra-level metadata (Figure 1). Each spectrum is composed of m/z values and associated intensities. The metadata are represented by a single table with variables along the columns and each row associated to a spectrum. Among the metadata available for each spectrum, there are MS level, acquisition number, retention time, precursor m/z and intensity (for MS level 2 and above), and many more. `MSnbase` relies on the `mzR` package[2] to import raw mass spectrometry data from one of the many community-maintained open standards formats (mzML, mzXML, mzData or ANDI-MS/netCDF) and provides a rich and principled interface to manipulate such objects. The code chunk below illustrates such an object as displayed in the R console and an enumeration of the metadata fields.

```
> show(ms)
MSn experiment data ("OnDiskMSnExp")
Object size in memory: 0.54 Mb
- - - Spectra data - - -
 MS level(s): 1 2 3
 Number of spectra: 994
 MSn retention times: 45:27 - 47:6 minutes
- - - Processing information - - -
```

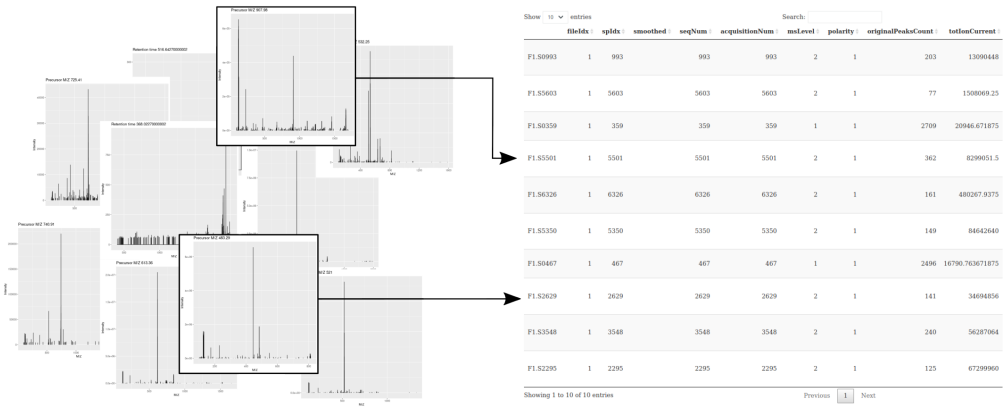

Figure 1: Schematic representation of what is referred to by *raw data*: a collection of mass spectra and a table containing spectrum-level annotations along the lines. Raw data are imported from one of the many community-maintained open standards formats (mzML, mzXML, mzData or ANDI-MS/netCDF).

```
Data loaded [Sun Apr 26 15:40:58 2020]
 MSnbase version: 2.13.6
- - - Meta data  - - -
phenoData
  rowNames: MS3TMT11.mzML
  varLabels: sampleNames
  varMetadata: labelDescription
Loaded from:
  MS3TMT11.mzML
protocolData: none
featureData
  featureNames: F1.S001 F1.S002 ... F1.S994 (994 total)
  fvarLabels: fileIdx spIdx ... spectrum (35 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
> fvarLabels(ms)
 [1] "fileIdx"                    "spIdx"
```

```
 [3] "smoothed"                    "seqNum"
 [5] "acquisitionNum"              "msLevel"
 [7] "polarity"                    "originalPeaksCount"
 [9] "totIonCurrent"               "retentionTime"
[11] "basePeakMZ"                  "basePeakIntensity"
[13] "collisionEnergy"             "ionisationEnergy"
[15] "lowMZ"                       "highMZ"
[17] "precursorScanNum"            "precursorMZ"
[19] "precursorCharge"             "precursorIntensity"
[21] "mergedScan"                  "mergedResultScanNum"
[23] "mergedResultStartScanNum"    "mergedResultEndScanNum"
[25] "injectionTime"               "filterString"
[27] "spectrumId"                  "centroided"
[29] "ionMobilityDriftTime"        "isolationWindowTargetMZ"
[31] "isolationWindowLowerOffset" "isolationWindowUpperOffset"
[33] "scanWindowLowerLimit"        "scanWindowUpperLimit"
[35] "spectrum"
```

In the following sections, we describe how `MSnbase` can be used for data processing and visualisation. An example of its ability to also efficiently handle very large mass spectrometry experiments (in this case with 5,773,464 spectra in 1,182 mzXML files) is provided as supplementary information. We will also illustrate how it makes use of the forward-pipe operator (`%>%`) defined in the `magrittr` package. This operator has proved useful to develop non-trivial analyses by combining individual functions into easily readable and elegant pipelines.

On-disk backend

The main feature in version 2 of the `MSnbase` package was the addition of different backends for raw data storage, namely *in-memory* and *on-disk*. The following code chunk demonstrates how to import data from an mzML file to create two `MSnExp` objects that store the data either in memory or on disk.

```
library("MSnbase")
raw_mem <- readMSData("file.mzML", mode = "inMemory")
raw_dsk <- readMSData("file.mzML", mode = "onDisk")
```

Both modes rely on the `mzR`[2] package to access the spectra (using the `mzR::peaks()` function) and the metadata (using the `mzR::header()` function) in the data files. The former is the legacy storage mode, implemented in the first version of the package, that loads all the raw data and the metadata into memory upon creation of the in-memory `MSnExp` object. This solution doesn't scale for modern large dataset, and was complemented by the on-disk backend. The on-disk backend only loads the metadata into memory when the on-disk `MSnExp` is created and accesses the spectra data (i.e. m/z and intensity values) in the original files on disk only when needed (see below and Figure 2 (d)), such as for example for plotting. There are two direct benefits using the on-disk backend, namely faster reading and reduced memory footprint. Figure 2 shows 5-fold faster reading times (a) and over a 10-fold reduction in memory usage (b).

Because the on-disk backend does not hold all the spectra data in memory, direct manipulations of these data are not possible. We thus implemented a *lazy processing* mechanism for this backend that caches any data manipulation operations in a processing queue in the object itself. These operations are then applied only when the user accesses m/z or intensity values. As an additional advantage, operations on subsets of the data become much faster since data manipulations are applied only to data subsets instead of the full data set at once. Also, on-disk data access is parallelized by data file ensuring a higher performance of this

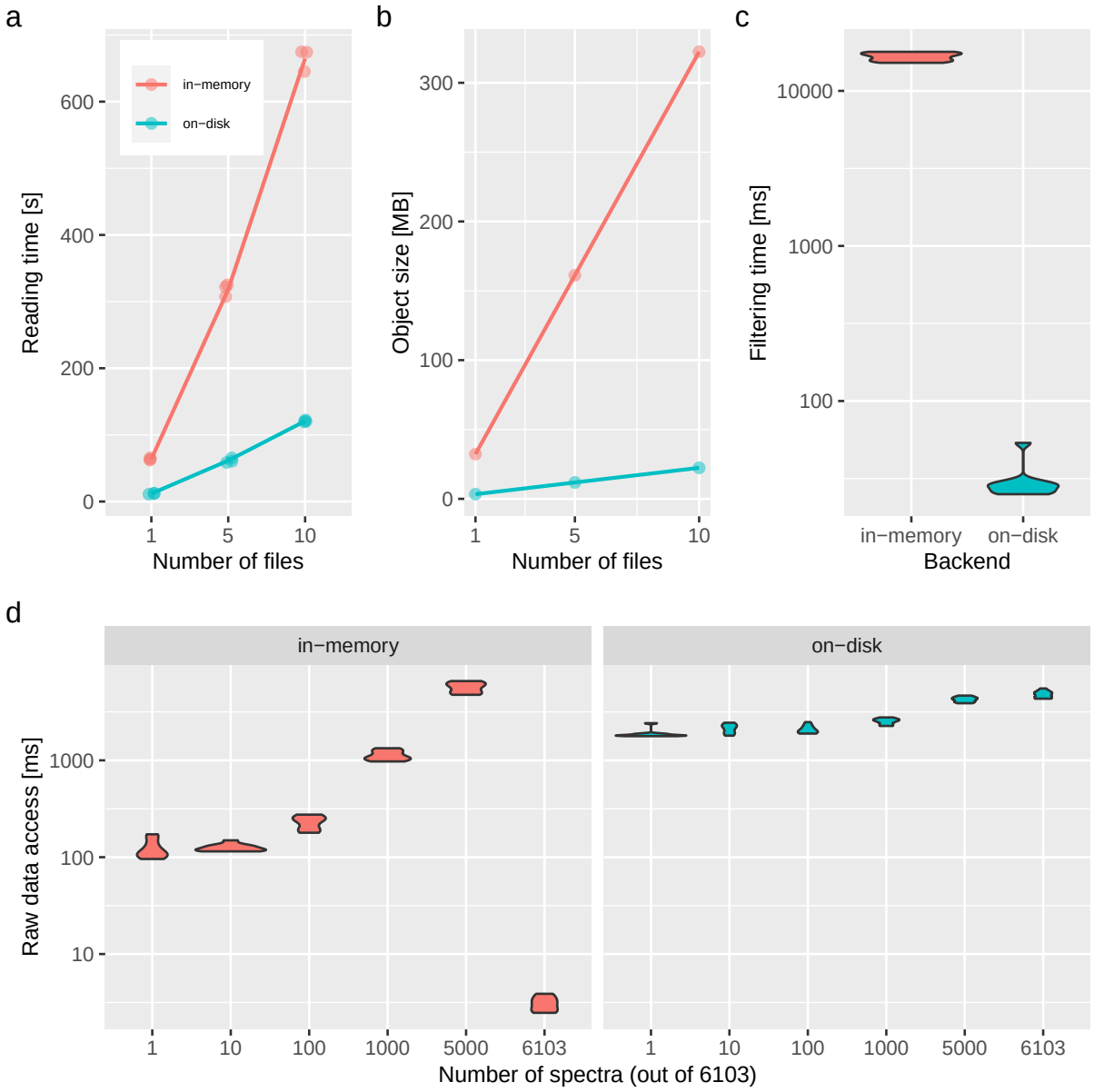

Figure 2: (a) Reading time (triplicates, in seconds) and (b) data size in memory (in MB) to read/store 1, 5 and 10 files containing 1431 MS1 (on-disk only) and 6103 MS2 (on-disk and in-memory) spectra. (c) Filtering benchmark assessed over 10 interactions on in-memory and on-disk data containing 6103 MS2 spectra. (d) Access time to spectra for the in-memory (left) and on-disk (right) backends for 1, 10, 100 1000, 5000 and all 6103 spectra. Benchmarks were performed on a Dell XPS laptop with an Intel i5-8250U processor 1.60 GHz (4 cores, 8 threads), 7.5 GB RAM running Ubuntu 18.04.4 LTS 64-bit and an SSD drive. The data used for the benchmarking are a TMT 4-plex experiment acquired on a LTQ Orbitrap Velos (Thermo Fisher Scientific) available in the `msdata` package and described in[14].

backend over conventional in-memory data representations. As an example, the following short analysis pipeline, that can equally be applied to in-memory or on-disk data, retains MS2 spectra acquired between 1000 and 3000 seconds, extracts the m/z range corresponding to the TMT 6-plex range and focuses on the MS2 spectra with a precursor intensity greater than 11×10^6 (the median precursor intensity).

```
ms <- ms %>%
    filterRt(c(1000, 3000)) %>%
    filterMz(120, 135)
ms[precursorIntensity(ms) > 11e6, ]
```

As shown on Figure 2 (c), this lazy mechanism is significantly faster than its application on in-memory data. The advantageous reading and execution times and memory footprint of the on-disk backend are possible by retrieving only spectra data from the selected subset hence avoiding access to the full raw data. Once access to the spectra m/z and intensity values become mandatory (for example for plotting), then the in-memory backend becomes more efficient, as illustrated on Figure 2 (d). The benefit of accessing data in memory is however reduced by underlying copies that are performed during the subsetting operation. When subsetting an in-memory `MSnExp` into a new, smaller in-memory `MSnExp` instance, the matrices that contain the spectra for the new object are copied, thus leading to increased execution time and (transient, if the original data are replaced) memory usage. Figure 2 (d) shows that the larger the subset, the smaller the benefits of an in-memory backend become. The example with the 6103 spectra, corresponding to the full data (i.e. all spectra are already in memory and there is no memory management overhead) is representative of memory access only and constitutes the best case scenario.

The on-disk backend has become the preferred backend for large data, and the only viable alternative when the size of the data exceeds the available RAM and/or when several MS levels are to be loaded and handled simultaneously. The in-memory backend can still prove

useful in cases when small MS2-only data are to be analysed, and will remain available in future versions of `MSnbase`.

Prototyping

The `MSnExp` data structure and its interface constitute an efficient prototyping environment for computational method development. We illustrate this by demonstrating how to implement the BoxCar[15] acquisition method. In a nutshell, BoxCar acquisition aims at improving the detection of intact precursor ions by distributing the charge capacity over multiple narrow m/z segments and thus limiting the proportion of highly abundant precursors in each segment. A full scan is reconstructed by combining the respective adjacent segments of the BoxCar acquisitions. The `MSnbaseBoxCar` package[16] is a small package that demonstrates this. The simple pipeline is composed of three steps, described below, and illustrated with code from `MSnbaseBoxCar` in the following code chunk.

1. Identify and filter the groups of spectra that represent adjacent BoxCar acquisitions (Figure 3 (b)). This can be done using the *filterString* metadata variable that identifies BoxCar spectra by their adjacent m/z segments with the `bc_groups()` function and filtering relevant spectra with the `filterBoxCar()`.

2. Remove any signal outside the BoxCar segments using the `bc_zero_out_box()` function from `MSnbaseBoxCar` (Figures 3 (c) and (d)).

3. Using the `combineSpectra` function from the `MSnbase`, combine the cleaned BoxCar spectra into a new, full spectrum (Figure 3 (e)).

```
bc <- readMSData("boxcar.mzML", mode = "onDisk") %>%
      bc_groups() %>%       ## identify BoxCar groups (creates 'bc_groups')
      filterBoxCar() %>%    ## keep only BoxCar spectra
      bc_zero_out_box() %>% ## remove signal outside of BoxCar segments
```

```
    combineSpectra(fcol = "bc_groups", ## reconstruct full spectrum
                   method = boxcarCombine)
```

After processing of the BoxCar data, the final object can either be further analysed using `MSnbase` or written back to disk as an `mzML` file using `writeMSData()` for processing with other tools.

All the functions for the processing of BoxCar spectra and segments in `MSnbaseBoxCar` were developed using existing functionality implemented in `MSnbase`, illustrating the flexibility and adaptability of the `MSnbase` package for computational mass spectrometry method development.

Visualisation

The R environment is well known for the quality of its visualisation capacity. This also holds true for mass spectrometry.[18–21] Here, we conclude the overview of version 2 of the `MSnbase` package by highlighting the flexibility of the software to visualise and assess the efficiency of raw data processing. Figure 4 compares the raw MS profile data imported from an mzML file for serine and the same data after smoothing, centroiding and m/z refinement, as illustrated in the code chunk below. Detailed execution and description of these operations can be found in the *MSnbase: centroiding of profile-mode MS data* `MSnbase` vignette.

```
serine_mz <- 106.049871
serine_proc <- ms %>%
    smooth(method = "SavitzkyGolay", halfWindowSize = 4L) %>%
    pickPeaks(refineMz = "descendPeak") %>%
    filterMz(c(serine_mz - 0.01, serine_mz + 0.01)) %>%
    filterRt(c(175, 187))
```

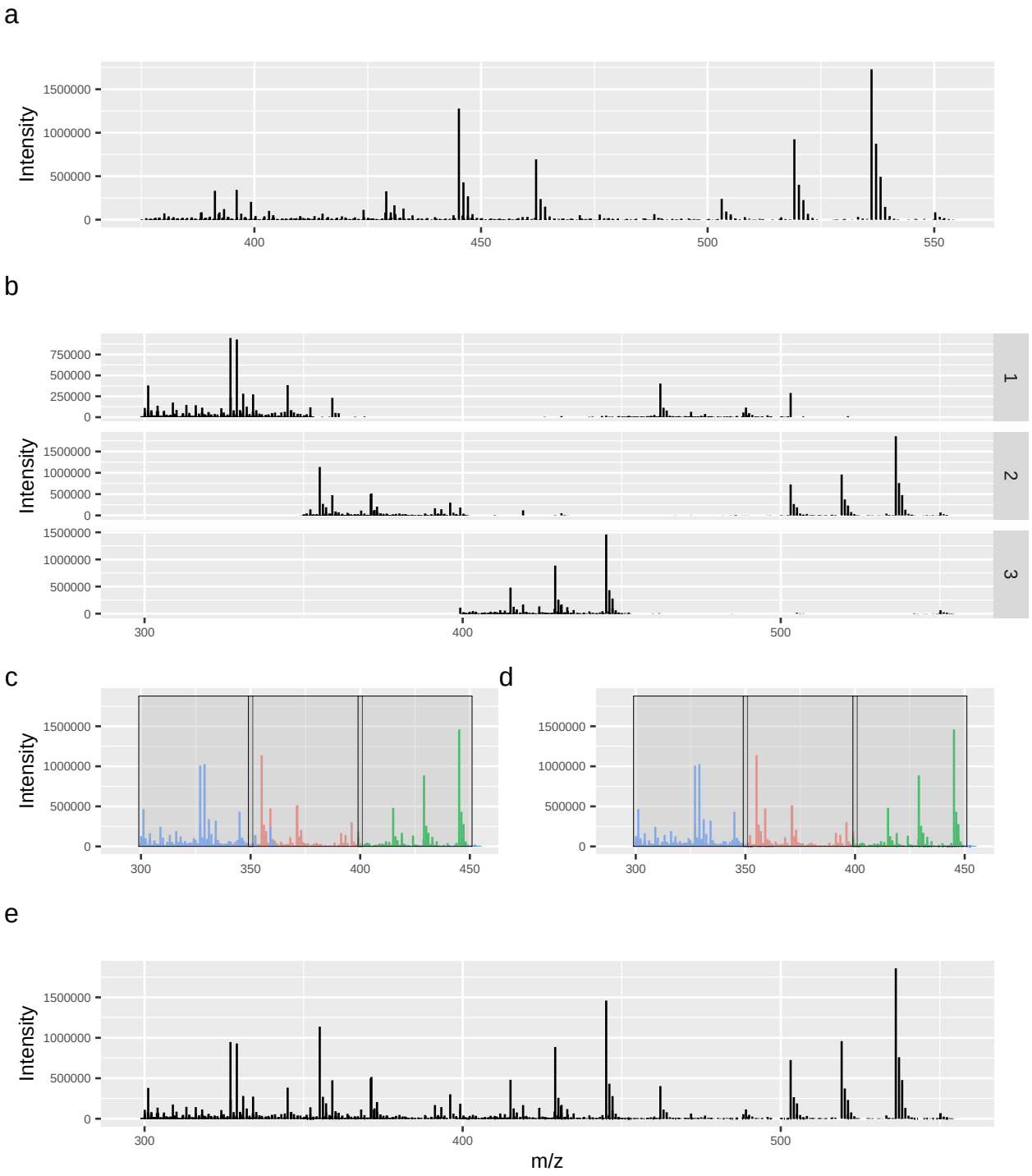

Figure 3: BoxCar processing with `MSnbase`. (a) Standard full scan with (b) three corresponding BoxCar scans showing the adjacent segments. Figure (c) shows the overlapping intact BoxCar segments and (d) the same segments after cleaning, i.e. where peaks outside of the segments were removed. The reconstructed full scan is shown on panel (e). Spectra visualisation, as shown here, rely on the `ggplot2`[17] package.

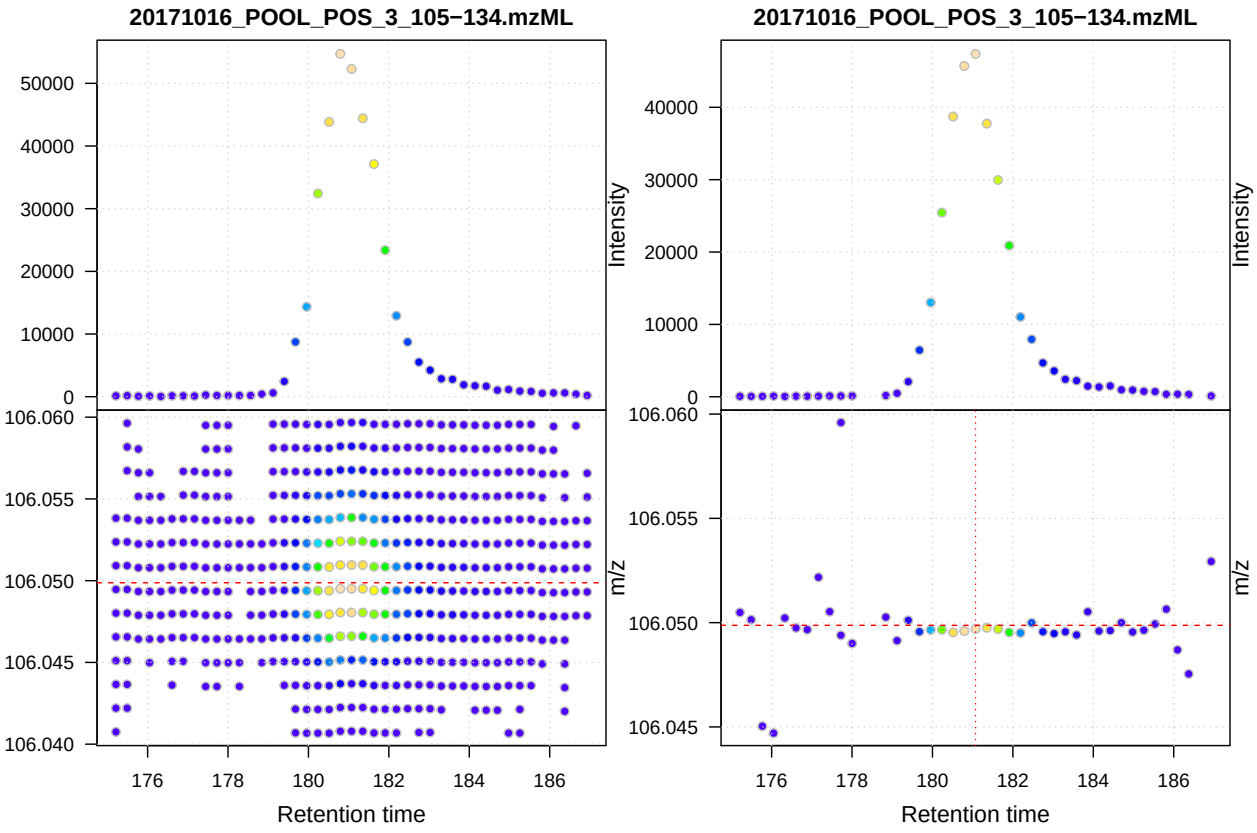

Figure 4: Visualisation of data smoothing and m/z refinement using `MSnbase`. (a) Raw MS profile data for serine. Upper panel shows the base peak chromatogram (BPC), lower panel the individual signals in the retention time – m/z space. The horizontal dashed red line indicates the theoretical m/z of the [M+H]+ adduct of serine. (b) Smoothed and centroided data for serine with m/z refinement. The horizontal red dashed line indicates the theoretical m/z for the [M+H]+ ion and the vertical red dotted line the position of the maximum signal.

Package maintenance and governance

The first public commit to the `MSnbase` GitHub repository was in October 2010. Since then, the package benefited from 12 contributors[22] that added various features, some particularly significant ones such as the on-disk backend described herein. Contributions to the package are explicitly encouraged, rewarded by an official contributor status and governed by a code of conduct.

According to `MSnbase`'s Bioconductor page, there are 36 Bioconductor packages that depend, import or suggest it. Among these are `pRoloc`[23] to analyse mass spectrometry-based spatial proteomics data, `msmsTests`,[24] `DEP`,[25] `DAPAR` and `ProStaR`[26] for the statistical analysis of quantitative proteomics data, `RMassBank`[27] to process metabolomics tandem MS files and build MassBank records, `MSstatsQC`[28] for longitudinal system suitability monitoring and quality control of targeted proteomic experiments and the widely used `xcms`[29] package for the processing and analysis of metabolomics data. `MSnbase` is also used in non-R/Bioconductor software, such as for example IsoProt,[30] that provides a reproducible workflow for iTRAQ/TMT experiments. The BioContainers[31] project offers a dedicated container for the `MSnbase` package, this facilitating the reuse of the package in third-party pipelines. `MSnbase` currently ranks 101 out of 1823 packages based on the monthly downloads from unique IP addresses, tallying over 1000 downloads from unique IP addresses each months.

As is custom with Bioconductor packages, `MSnbase` comes with ample documentation. Every user-accessible function is documented in a dedicated manual page. In addition, the package offers 5 vignettes, including one aimed at developers. The package is checked nightly on the Bioconductor servers: it implements unit tests covering 72% of the code base and, through its vignettes, also provides integration testing. Questions from users and developers are answered on the Bioconductor support forum as well as on the package GitHub page. The package provides several sample and benchmarking datasets, and relies on other dedicated *experiment packages* such as `msdata`[32] for raw data or `pRolocdata`[23] for quantitative data.

`MSnbase` is available on Windows, Mac OS and Linux under the open source Artistic 2.0 license and easily installable using standard installation procedures.

The growth of `MSnbase` and the user support provided over the years attest to the core maintainers commitment to long-term development, and the quality and maintainability of the code base.

Discussion

We have presented here some important functionality of `MSnbase` version 2. The new on-disk infrastructure enables large scale data analyses,[33] either using `MSnbase` directly or through packages that rely on it, such as `xcms`. We have also illustrated how `MSnbase` can be used for standard data analysis and visualisation, and how it can be used for method development and prototyping.

The version of `MSnbase` used in this manuscript is 2.14.2. The main features presented here were available since version 2.0. The code to reproduce the analyses and figures in this article is available at `https://github.com/lgatto/2020-msnbase-v2/`.

Associated Content

Supplementary file 1: script documenting the processing of 1182 mzXML files (5773464 spectra) using `MSnbase`.

Acknowledgement

The authors thank the various contributors and users who have provided constructive input and feedback that have helped, over the years, the improvement of the package. The authors declare no conflict of interest.

References

(1) Côté Richard G, M. L., Reisinger Florian jmzML, an open-source Java API for mzML, the PSI standard for MS data. *Proteomics* **2010**, *10*, 1332–1335.

(2) Chambers, M. C. et al. A cross-platform toolkit for mass spectrometry and proteomics. *Nat Biotechnol* **2012**, *30*, 918–20.

(3) Cox, J.; Mann, M. MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification. *Nat Biotechnol* **2008**, *26*, 1367–1372.

(4) Vaudel, M.; Burkhart, J. M.; Zahedi, R. P.; Oveland, E.; Berven, F. S.; Sickmann, A.; Martens, L.; Barsnes, H. PeptideShaker enables reanalysis of MS-derived proteomics data sets. *Nat Biotechnol* **2015**, *33*, 22–24.

(5) Gatto, L.; Lilley, K. S. `MSnbase` - an R/Bioconductor package for isobaric tagged mass spectrometry data visualization, processing and quantitation. *Bioinformatics* **2012**, *28*, 288–9.

(6) Huber, W. et al. Orchestrating high-throughput genomic analysis with Bioconductor. *Nat Methods* **2015**, *12*, 115–21.

(7) Rost, H. L.; Schmitt, U.; Aebersold, R.; Lars, M. pyOpenMS: a Python-based interface to the OpenMS mass-spectrometry algorithm library. *Proteomics* **2014**, *14*, 74–77.

(8) Bittremieux, W. spectrum_utils: A Python package for mass spectrometry data processing and visualization. *Analytical Chemistry* **2020**, *92*, 659–661.

(9) Goloborodko, A. A.; Levitsky, L. I.; Ivanov, M. V.; Gorshkov, M. V. Pyteomics - a Python Framework for Exploratory Data Analysis and Rapid Software Prototyping in Proteomics. *J. Am. Soc. Mass Spectrom.* **2013**, *24*, 301–304.

(10) R Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing: Vienna, Austria, 2020.

(11) Stanstrup, J. et al. The metaRbolomics Toolbox in Bioconductor and Beyond. *Metabolites* **2019**, *9*, 200.

(12) Gatto, L. `MSnbase` Base Functions and Classes For Mass Spectrometry and Proteomics. 2020; R package version 2.14.2.

(13) Rainer, J. Metabolomics data pre-processing using `xcms`. 2020.

(14) Gatto, L.; Christoforou, A. Using R and Bioconductor for proteomics data analysis. *Biochim Biophys Acta* **2014**, *1844*, 42–51.

(15) Meier, F.; Geyer, P.; Virreira Winter, S.; Juergen, C.; Matthias, M. BoxCar acquisition method enables single-shot proteomics at a depth of 10,000 proteins in 100 minutes. *Nature Methods* **2018**, *15*, 440–448.

(16) Gatto, L. `MSnbaseBoxCar`: BoxCar Data Processing with MSnbase. 2020; R package version 0.2.0.

(17) Wickham, H. *ggplot2: Elegant Graphics for Data Analysis*; Springer-Verlag New York, 2016.

(18) Gatto, L.; Breckels, L. M.; Naake, T.; Gibb, S. Visualization of proteomics data using R and Bioconductor. *PROTEOMICS* **2015**, *15*, 1375–1389.

(19) Panse, C.; Grossmann, J. `protViz`: Visualizing and Analyzing Mass Spectrometry Related Data in Proteomics. 2020; R package version 0.6.

(20) Sauteraud, R.; Jiang, M.; Gottardo, R. `Pviz`: Peptide Annotation and Data Visualization using Gviz. 2019; R package version 1.21.0.

(21) Bemis, K. D.; Harry, A.; Eberlin, L. S.; Ferreira, C.; van de Ven, S. M.; Mallick, P.; Stolowitz, M.; Vitek, O. `Cardinal`: an R package for statistical analysis of mass spectrometry-based imaging experiments. *Bioinformatics* **2015**,

(22) Gatto, L. `MSnbase` contributors 2010 - 2020. 2020; `https://lgatto.github.io/msnbase-contribs-2/`.

(23) Gatto, L.; Breckels, L. M.; Wieczorek, S.; Burger, T.; Lilley, K. S. Mass-spectrometry-based spatial proteomics data analysis using `pRoloc` and `pRolocdata`. *Bioinformatics* **2014**, *30*, 1322–4.

(24) Gregori, J.; Sanchez, A.; Villanueva, J. msmsTests: LC-MS/MS Differential Expression Tests. 2019; R package version 1.25.0.

(25) Zhang, X.; Smits, A. H.; van Tilburg, G. B.; Ovaa, H.; Huber, W.; Vermeulen, M. Proteome-wide identification of ubiquitin interactions using UbIA-MS. *Nature Protocols* **2018**, *13*, 530–550.

(26) Wieczorek, S.; Combes, F.; Lazar, C.; Giai Gianetto, Q.; Gatto, L.; Dorffer, A.; Hesse, A. M.; Couté, Y.; Ferro, M.; Bruley, C.; Burger, T. `DAPAR` & `ProStaR`: software to perform statistical analyses in quantitative discovery proteomics. *Bioinformatics* **2017**, *33*, 135–136.

(27) Stravs, M. A.; Schymanski, E. L.; Singer, H. P.; Hollender, J. Automatic Recalibration and Processing of Tandem Mass Spectra using Formula Annotation. *Journal of Mass Spectrometry* **2013**, *48*, 89–99.

(28) Dogu, E.; Mohammad-Taheri, S.; Abbatiello, S. E.; Bereman, M. S.; MacLean, B.; Schilling, B.; Vitek, O. `MSstatsQC`: Longitudinal System Suitability Monitoring and Quality Control for Targeted Proteomic Experiments. *Mol Cell Proteomics* **2017**, *16*, 1335–1347.

(29) Smith, C. A.; Want, E. J.; O'Maille, G.; Abagyan, R.; Siuzdak, G. XCMS: processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching, and identification. *Anal Chem* **2006**, *78*, 779–87.

(30) Griss, J.; Vinterhalter, G.; Schwämmle, V. IsoProt: A Complete and Reproducible Workflow To Analyze iTRAQ/TMT Experiments. *J Proteome Res* **2019**, *18*, 1751–1759.

(31) da Veiga Leprevost, F. et al. BioContainers: an open-source and community-driven framework for software standardization. *Bioinformatics* **2017**, *33*, 2580–2582.

(32) Neumann, S.; Gatto, L.; Rainer, J. `msdata`: Various Mass Spectrometry raw data example files. 2019; R package version 0.27.0.

(33) Nothias, L. F. et al. Feature-based molecular networking in the GNPS analysis environment. *Nat Methods* **2020**,

Supplementary data for: 'MSnbase', efficient and elegant R-based processing and visualisation of raw mass spectrometry data

Laurent Gatto, Sebastian Gibb and Johannes Rainer

4 August 2020

Contents

1 Introduction 2

2 Data handling and analysis with `MSnbase` 2

2.1 Basic MS data access functionality 3

2.2 Performance of the on-disk backend on large scale data sets 4

2.3 System information 7

Supplementary data for: 'MSnbase', efficient and elegant R-based processing and visualisation of raw mass spectrometry data

1 Introduction

This document describes handling of mass spectrometry data from large experiments using the `MSnbase` package and more specifically its *on-disk* backend. For demonstration purposes, the MassIVE data set MSV000080030 is used. This consists of over 1,000 mzXML files from swab-samples collected from hands and various personal objects of 80 volunteers.

2 Data handling and analysis with `MSnbase`

In this section we demonstrate data handling and access by `MSnbase` on a large experiment consisting of more than 1,000 data files.

To reproduce the analysis described in this document, download the *MSV000080030* folder from ftp://massive.ucsd.edu/MSV000080030/ and place it into the same folder as this document.

Below we load the required libraries and define the files to be analyzed.

```
library(MSnbase)
library(magrittr)
library(pryr)

fls <- dir("MSV000080030/ccms_peak/Forensic_study_80_volunteers/",
           pattern = "mzXML", full.names = TRUE, recursive = TRUE)
```

The data set consists of 1182 mzXML files. We next load the data using the two different `MSnbase` backends `"inMemory"` and `"onDisk"`. For the in-memory backend, due to the larger memory requirements, we import the data only from a subset of the files.

```
ms_mem <- readMSData(fls[grep("Hand", fls)], mode = "inMemory")
```

Next we load data from all mzXML files as an on-disk `MSnExp` object.

```
ms_dsk <- readMSData(fls, mode = "onDisk")
```

Below we count the number of spectra per MS level of the whole experiment.

```
table(msLevel(ms_dsk))
##
##       1       2
## 1173678 4599786
```

Note that the in-memory `MSnExp` object contains only MS2 spectra (in total 2140520) from a subset of data files. However, the data import was much slower (over ~ 12 hours for the in-memory backend while creating the on-disk object from the full data data set took ~ 3 hours).

Next we subset the on-disk object to contain the same set of spectra as the in-memory `MSnExp` and compare their memory footprint.

```
ms_dsk_hands <- ms_dsk %>%
    filterFile(grep("Hand", fls)) %>%
    filterMsLevel(2L)
```

```
object_size(ms_mem)
## 21.8 GB
object_size(ms_dsk_hands)
## 617 MB
```

Since the on-disk object stores only spectra metadata in memory it occupies also much less system memory. As a comparison, the on-disk `MSnExp` for the full experiment was still much smaller than the in-memory object:

```
object_size(ms_dsk)
## 1.66 GB
```

2.1 Basic MS data access functionality

Before evaluating the `MSnbase` performance on the large data set we provide some general description of the `MSnbase` data classes and basic data access operations. MS data from raw data files in mzML, mzXML, mzData or netCDF format is represented by the `MSnExp` object which organizes the spectra from the original files in an one-dimensional list. Functions like `rtime` and `msLevel` allow to extract the retention time and MS level, respectively. They return a `numeric` (or `integer`) vector with the same length as the number of spectra in the `MSnExp`. In the example below we use the `rtime` function to extract the retention times for each spectrum.

```
rts <- rtime(ms_dsk)
length(rts)
## [1] 5773464
head(rts)
## F0001.S0001 F0001.S0002 F0001.S0003 F0001.S0004 F0001.S0005 F0001.S0006
##       0.470       0.803       1.136       1.468       1.801       2.134
```

The `fromFile` function can be used to determine the source file (sample) of a specific spectrum in the `MSnExp` object. This function returns an `integer` vector, of the same length as spectra in the experiment, with the file index. The file names can be accessed with the `fileNames` method. An `MSnExp` object can be subsetted with `[` and e.g. the index of the spectra that should be retained. In the code block below we subset our `ms_dsk` object to keep only spectra from the 3rd file.

```
one_file <- ms_dsk[fromFile(ms_dsk) == 3]
length(one_file)
## [1] 4911
```

Note that there are also dedicated *filter* functions to subset an `MSnExp` object such as `filterFile`, `filterMsLevel`, `filterRt`, `filterMz`, `filterPrecursorMz` or `filterIsolationWindow`. In the example below we use the `filterRt` function to further subset our data to keep only spectra acquired within a certain time range.

```
one_file <- filterRt(one_file, rt = c(40, 300))
length(one_file)
## [1] 1996
```

As mentioned above, the `MSnExp` object is comparable with a list of spectra. Thus, to extract a single spectrum from it we can use `[[`. This will return an object of type `Spectrum` which encapsules/represents all information of the measured spectrum (i.e. m/z and intensity values as well as metadata information). In the example below we extract the 15th spectrum from our data subset and access its m/z values with the `mz` function.

```
sp <- one_file[[15]]
mz(sp)
## [1]  400.4412  431.2400 1617.8282
```

This particular spectrum has only 3 peaks.

Note that m/z or intensity values can also be directly extracted from the `MSnExp` object as shown in the example below. The result will be a `list` of `numeric` vectors, each element representing the m/z values for each spectrum in the object.

```
mzs <- mz(one_file)
class(mzs)
## [1] "list"
length(mzs)
## [1] 1996
```

In addition, it is also possible to extract all m/z and intensity values from an `MSnExp` object as a `data.frame` as shown in the code block below, but this is not suggested, since it loads all the data into memory but all MS spectrum metadata such as MS level or precursor m/z get lost.

```
df <- as(one_file, "data.frame")
head(df)
##   file     rt       mz   i
## 1    1 40.118 387.2650  88
## 2    1 40.118 389.2627 192
## 3    1 40.118 474.2964 164
## 4    1 40.450 387.2416 212
## 5    1 40.450 389.2666 184
## 6    1 40.450 445.2680 132
nrow(df)
## [1] 2854657
```

Note that for all these operations it is irrelevant whether an in-memory or on-disk backend was used. In general it is advisable to use the on-disk backend especially for experiments with more than ~ 50 files.

2.2 Performance of the on-disk backend on large scale data sets

To demonstrate `MSnbase`'s efficiency in processing large scale experiments we perform some standard subsetting, data access and manipulation operations.

We first compare the performance of the on-disk and in-memory backend on accessing m/z values with the `mz` function on a set of 100 randomly selected spectra. The performance is assessed with the `microbenchmark` function.

```
set.seed(123)
idx <- sample(seq_along(ms_mem), 100)

library(microbenchmark)
microbenchmark(mz(ms_mem[idx]),
               mz(ms_dsk_hands[idx]),
               times = 5)
## Unit: seconds
##                   expr      min        lq     mean   median       uq      max
##        mz(ms_mem[idx]) 46.77433 47.461748 47.80530 47.92476 48.37474 48.49095
##  mz(ms_dsk_hands[idx])  3.58289  3.636521 12.96023  3.64256 26.93021 27.00896
##  neval
##      5
##      5
```

For this combined subsetting and data access operation the on-disk backend performed better than the in-memory `MSnExp`, while even requiring much less memory.

Next we extract all MS2 spectra with a retention time between 50 and 60 seconds and a precursor m/z of 108.5362 (+/- 5ppm). This subsetting operation is performed on the on-disk `MSnExp` object representing the full experiment with the 1182 data files/samples. To assess the performance of the following operations we use `system.time` calls that record elapsed time in seconds.

```
system.time(
    ms_sub <- ms_dsk %>%
        filterMsLevel(2L) %>%
        filterRt(c(50, 60)) %>%
        filterPrecursorMz(mz = 108.5362, ppm = 5)
)["elapsed"]
## elapsed
##   4.571
```

In total `length(ms_sub)` spectra were selected from in total 928 data files/samples. The plot below shows the data for the first spectrum.

```
system.time(
    plot(ms_sub[[1]])
)["elapsed"]
```

```
## elapsed
##   0.291
```

Since there seems to be quite some background noise in the MS2 spectrum we next remove peaks with an intensity below 50 by first replacing their intensities with 0 (with the `removePeaks` call) and subsequently removing all 0-intensity peaks from each spectrum with the `clean` call. In addition we *normalize* each spectrum by dividing the maximum intensity per spectrum from the spectrum's intensities.

```
system.time(
    ms_sub <- ms_sub %>%
        removePeaks(t = 50) %>%
        clean(all = TRUE) %>%
```

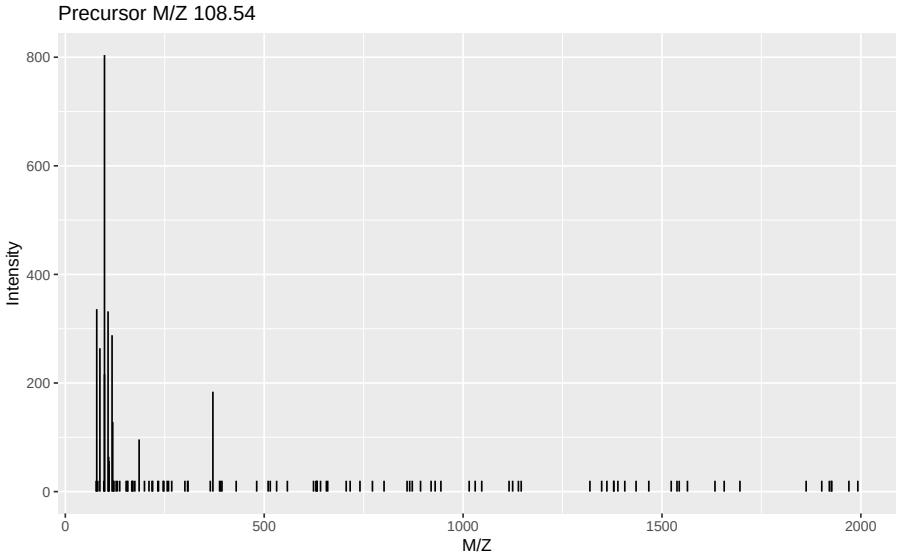

Figure S1: Example spectrum of the data set

```
        normalize(method = "max")
)["elapsed"]
## elapsed
##   0.006
```

The result on the first spectrum is shown below.

```
system.time(
    plot(ms_sub[[1]])
)["elapsed"]
```

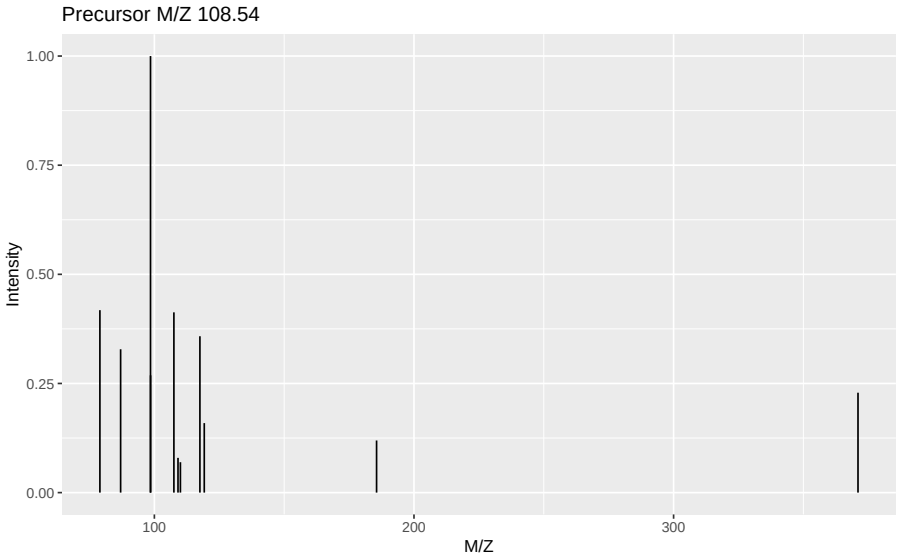

Figure S2: Example spectrum after cleaning

```
## elapsed
##   0.161
```

Note that any of the data manipulations above are not directly applied to the data but *cached* in the object's internal *lazy processing queue* (explaining the very short running time of the normalization call). The operations are only effectively applied to the data when m/z or intensity values are extracted from the object, e.g. in the `plot` call above.

For additional workflows employing `MSnbase` see also metabolomics2018[1] that explains filtering, plotting and centroiding of profile-mode MS data with `MSnbase` and subsequent pre-processing of the (label free/untargeted) LC-MS data with the `xcms` package (that builds upon `MSnbase` for MS data representation and access).

[1] https://github.com/jorainer/metabolomics2018

2.3 System information

The present analysis was run on a MacBook Pro 16,1 with 2.3 GHz 8-Core Intel Core i9 CPU and 64 GB 2667 MHz DDR4 memory running macOS version 10.15.5. The R version and the version of the used packages are listed below.

```
sessionInfo()
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.5
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] microbenchmark_1.4-7 BiocParallel_1.22.0  pryr_0.1.4
##  [4] magrittr_1.5         MSnbase_2.14.2       ProtGenerics_1.20.0
##  [7] S4Vectors_0.26.1     mzR_2.22.0           Rcpp_1.0.5
## [10] Biobase_2.48.0       BiocGenerics_0.34.0  BiocStyle_2.16.0
## [13] rmarkdown_2.3
##
## loaded via a namespace (and not attached):
##  [1] tinytex_0.25          tidyselect_1.1.0      xfun_0.16
##  [4] purrr_0.3.4           lattice_0.20-41       colorspace_1.4-1
##  [7] vctrs_0.3.2           generics_0.0.2        htmltools_0.5.0
## [10] yaml_2.2.1            vsn_3.56.0            XML_3.99-0.5
## [13] rlang_0.4.7           pillar_1.4.6          glue_1.4.1
## [16] affy_1.66.0           foreach_1.5.0         affyio_1.58.0
## [19] lifecycle_0.2.0       plyr_1.8.6            mzID_1.26.0
## [22] stringr_1.4.0         zlibbioc_1.34.0       munsell_0.5.0
```

```
## [25] pcaMethods_1.80.0     gtable_0.3.0          codetools_0.2-16
## [28] evaluate_0.14         labeling_0.3          knitr_1.29
## [31] IRanges_2.22.2        doParallel_1.0.15     preprocessCore_1.50.0
## [34] scales_1.1.1          BiocManager_1.30.10   limma_3.44.3
## [37] farver_2.0.3          impute_1.62.0         ggplot2_3.3.2
## [40] digest_0.6.25         stringi_1.4.6         bookdown_0.20
## [43] dplyr_1.0.1           ncdf4_1.17            grid_4.0.2
## [46] tools_4.0.2           tibble_3.0.3          crayon_1.3.4
## [49] pkgconfig_2.0.3       ellipsis_0.3.1        MASS_7.3-51.6
## [52] iterators_1.0.12      R6_2.4.1              MALDIquant_1.19.3
## [55] compiler_4.0.2
```