Hands-on the Detection of Isotope Pattern of Mass Spectrometric Measurements

Christian Panse, Lucas Schmidt, Witold E. Wolski

Abstract

The CRAN package **deisotoper** provides a low-level interface for a deisotoper container implemented in the 'Java' programming language and means of S3 helper functions for plotting and debugging isotopes of mass spectrometric data. The implemented feature-based algorithm detects and aggregates peaks which belong to the same isotopic cluster of a given mass spectrum. One feature of the algorithm is that it can handle overlapping clusters.

Keywords: proteomics, mass spectrometry.

1. Preliminary notes

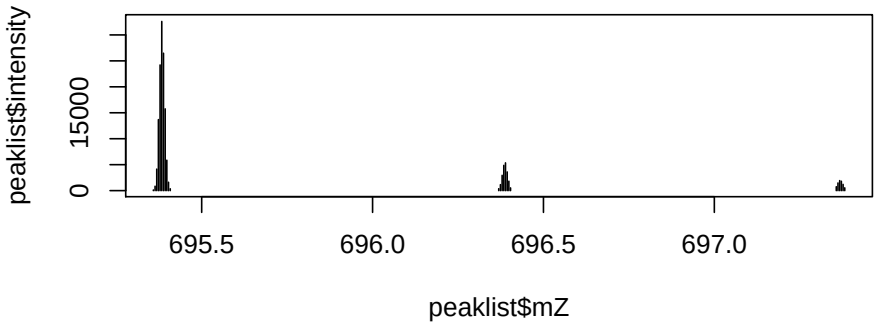
One chemical element can have different atomic masses caused by the different number of neutrons. The elementary mass of one neutron is approximately 1 Dalton. The chemical elements hydrogen, carbon, oxygen, nitrogen, and sulfate composing chemical structures known as amino acids. The amino acids are the building blocks for the construction of protein sequences. All these chemical structures are carrying the different number of neutron distributions of the containing elemental composition. In the area of proteomics mass spectrometry is the method of choice. Peptides, digested proteins, are measured in a mass spectrometric device. All mass spectra are queried to a set of in-silico computed peptide spectra. This process is usually done by using a tandem mass spectrometry (MS/MS) sequence database search software as mascot Perkins, Pappin, Creasy, and Cottrell (1999) or comet Eng, Hoopmann, Jahan, Egertson, Noble, and MacCoss (2015). De-isotope mass spectra have an impact on the score of the peptide-spectrum match (PSM) of the search algorithm. A Filtered and convoluted mass spectrum leads to a higher coverage between fragment ions and in-silico computed ions and results in a better score. Here we implemented and tested the feature-based algorithm described in Yuan, Shi, Lin, Chen, and Wu (2011).

In the following sections, we will demonstrate the usage of the **deisotoper** package.

2. The input

If the mass spec software delivers profile data, the profiles have to be fitted, e.g., by a Gaussian curve fitting and the expectation value has to be determined. In this section, we briefly demonstrate for the sake of completeness how that so-called peak picking preprocessing step can be handled using R. The plot below displays an isotope cluster in measured profile mode.

```
R> plot(peaklist$mZ, peaklist$intensity, type = 'h')
```

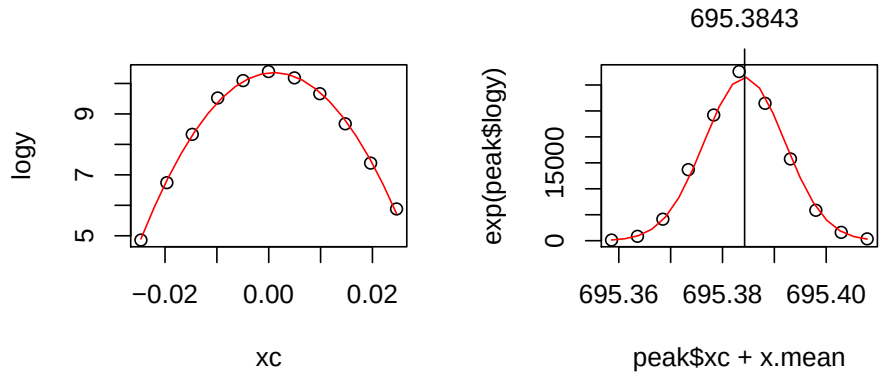


For demonstration purpose only we use the `lm` function for the fit of the first isotope. Therefore we apply a `log` transformation the Gaussian distribution equation.

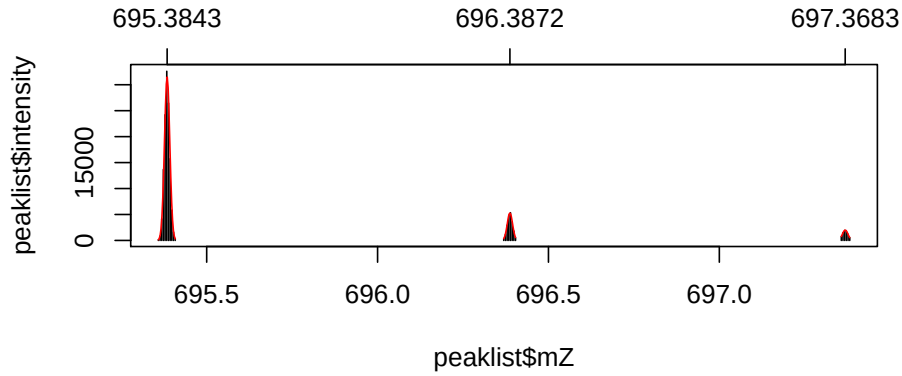
```
R> isotope1 <- 1:11
R> mean(peaklist$mZ[isotope1])

[1] 695.3832

R> x <- peaklist$mZ[isotope1]
R> y <- peaklist$intensity[isotope1]
R> peak <- data.frame(logy=log(y), x=x)
R> x.mean <- mean(peak$x)
R> peak$xc <- peak$x - x.mean
R> (fit <- lm(logy ~ xc + I(xc^2), data= peak))

Call:
lm(formula = logy ~ xc + I(xc^2), data = peak)

Coefficients:
(Intercept)           xc      I(xc^2)
      10.35        16.84     -8292.75
```

The plots below display the quadratic curve (left) and the predicted values of the model. The right figure shows the fitted Gaussian curve. Please note, the x-axis transformation by `x.mean` is applied to achieve a better numerical conditioning of the problem.

```
R> op <- par(mfrow=c(1,2))
R> plot(logy ~ xc, data=peak)
R> xx <- with(peak, seq(min(xc), max(xc), length=20))
R> lines(xx , predict(fit, data.frame(xc=xx)), col=2)
R> plot(peak$xc + x.mean, exp(peak$logy))
R> lines(xx + x.mean, exp(predict(fit, data.frame(xc = xx))), col=2)
R> x0 <- -fit$coefficients[2] / (2 *fit$coefficients[3])
R> abline(v=x0 + x.mean)
R> axis(3, x0 + x.mean, round(x0 + x.mean,4))
```



The graphics below show the fits (red curve) and expectation values (picked peaks) as tick-marks for all three isotopes of the isotopic cluster.



Nevertheless, today's mass spec devices are usually able to deliver the data in centroid mode.

3. The package usage

3.1. Input example 1: in-silico

```
R> library(deisotoper)
```

Lets define a blackboard example to demonstrate the "deisotoping" algorithm on two overlapping isotope clusters.

```
R> x0 <- list(mZ = c(1, 2, 2.5, 3), intensity = rep(1, 4),
+             pepmass=600, charge=2)
R> x1 <- list(mZ = c(1.01, 2, 2.5, 3), intensity = rep(1, 4),
+             pepmass=600, charge=2)
```

`deisotoper()` returns a reference to a standard configurated deisotoper object.

```
R> dtoper <- deisotoper()
```

The following line calls the deisotoper method of the object.

```
R> (xd0 <- deisotope(dtoper, x0))

$title
NULL

$rtinseconds
NULL

$charge
[1] 2

$scan
NULL

$pepmass
[1] 600

$mZ
[1] 1.0 2.5

$intensity
[1] 3 1

$id
NULL
```

```
R> print(dtoper)
```

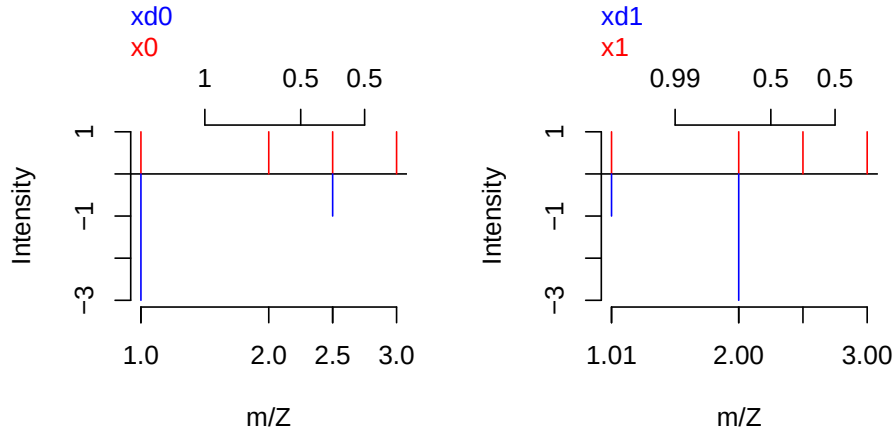
```
[1] IsotopicSet     IsotopicCluster Peak            Charge
[5] mZ              Intensity
<0 rows> (or 0-length row.names)
```

```
R> (xd1 <- deisotope(dtoper, x1))
```

```
$title
NULL

$rtinseconds
NULL

$charge
[1] 2

$scan
NULL

$pepmass
[1] 600

$mZ
[1] 1.01 2.00

$intensity
[1] 1 3

$id
NULL
```

plot both spectra

```
R> op <- par(mfrow=c(1,2))
R> plot.deisotoper(x0, xd0)
R> plot.deisotoper(x1, xd1)
```



The following line displays the content of the deisotoper object.

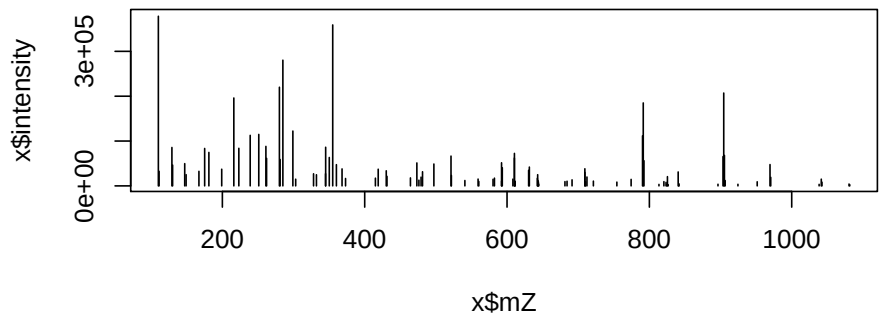
```
R> print(dtoper)
```

```
  IsotopicSet IsotopicCluster Peak Charge   mZ Intensity
1          -1              -1    0     -1 1.01         1
```

3.2. Input example 2: tandem mass spectrum

Given: A centroid mode generated mass spectrum.

```
R> plot(x$mZ, x$intensity, type='h')
```



Sanity check

```
R> eps <- 0.025
R> rv <- (lapply(1:3, function(charge){
```

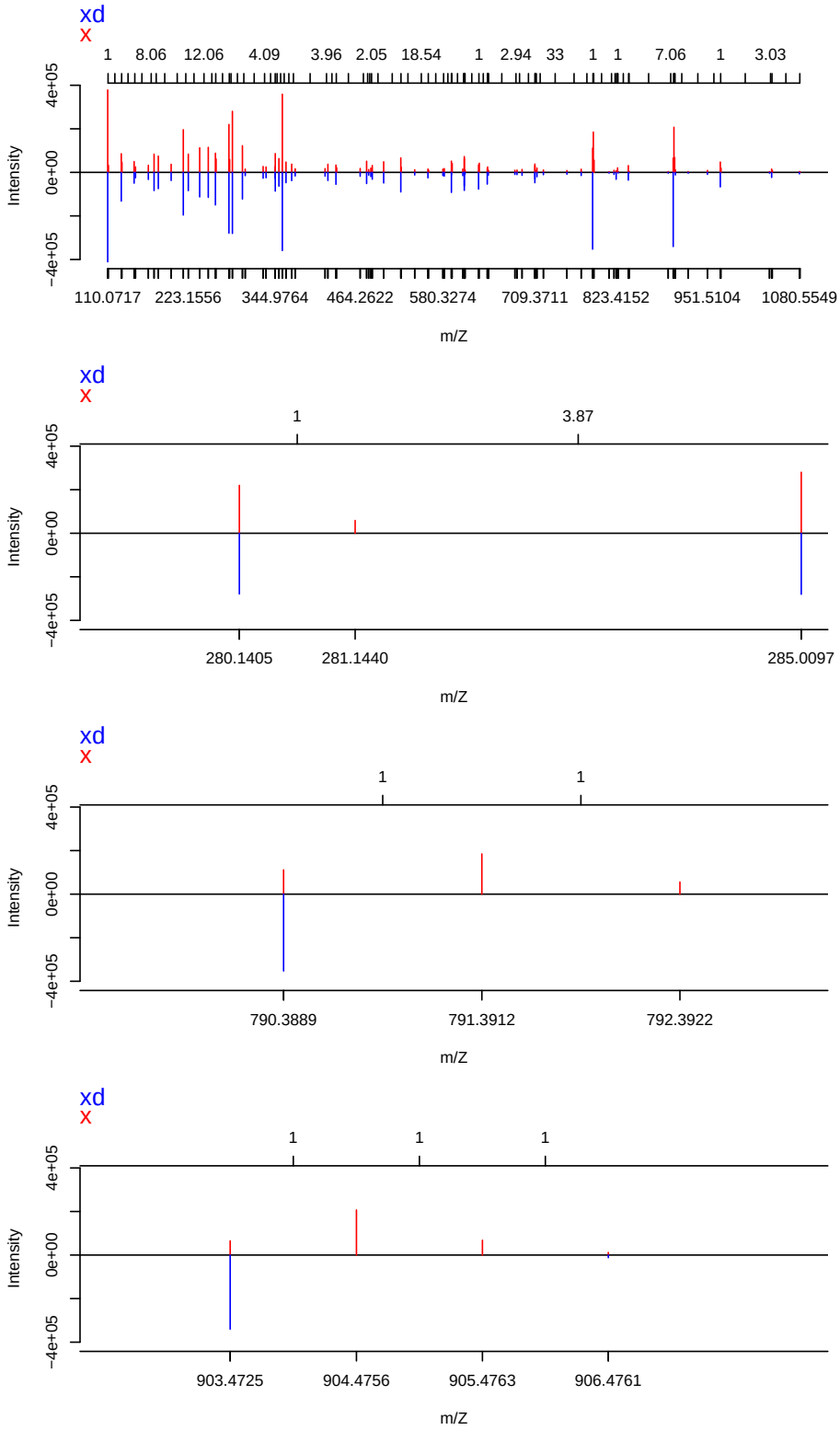
```
+    diff.mZ <- diff(x$mZ)
+    idx <- which( (1 / charge - eps) < diff.mZ
+                  & diff.mZ  < (1 / charge + eps))
+
+    diff.mZ[idx] - (1 / charge)
+  }))
R> rv
```

```
[[1]]
 [1]  0.00332 -0.01600  0.00345  0.00367  0.00445 -0.01702  0.00391
 [8]  0.00507  0.00232  0.00097 -0.00976 -0.01123  0.00311  0.00312
[15]  0.00067 -0.00025  0.00281  0.00537 -0.00720

[[2]]
[1]  0.00024 -0.01800  0.01500 -0.00700 -0.00500

[[3]]
numeric(0)
```

```
R> # standart configurated deisotoper
R> dtoper <- deisotoper(delta = 0.15)
R> # return the configuration of dtoper
R> config <- config.deisotoper(dtoper)
R> # deisotope the data
R> xd <- deisotope(dtoper, x)
R> summary.deisotoper(dtoper)
```

```
  NumberOfIsotopicSets NumberOfIsotopicClusters
1                   25                       29
  NumberOfPeaksInIsotopicClusters NumberOfPeaks
1                              51            98
```

```
R> # plot the example data and the deisotoped data
R> op <- par(mfrow=c(4, 1))
R> plot.deisotoper(x, xd)
R> plot.deisotoper(x, xd, xlim = c(279, 285))
R> plot.deisotoper(x, xd, xlim = c(789.5, 793))
R> plot.deisotoper(x, xd, xlim = c(902.5, 908))
R> par(op)
```

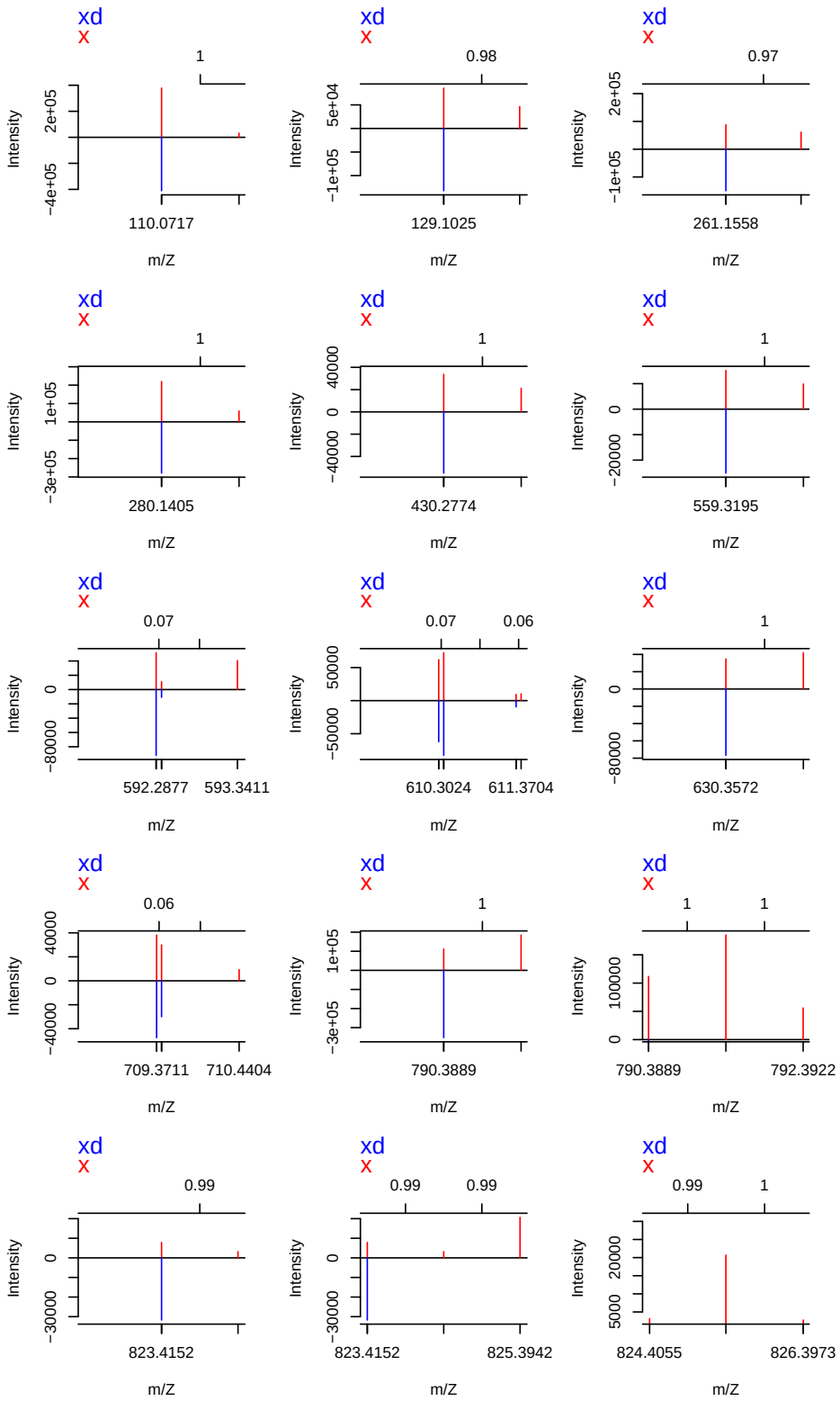



```
R> charge <- 1;
R> (eps <- as.numeric(
```

```
+    as.character(config$Value[which(config$Configuration == "Delta")])))
```

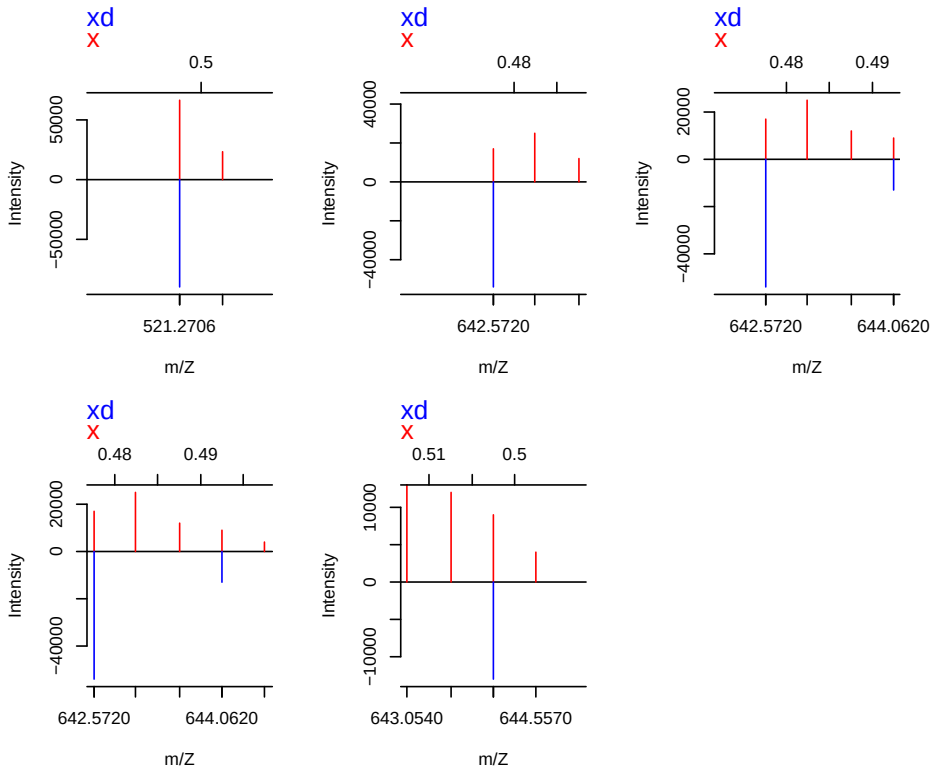
```
[1] 0.15
```

```
R> diff.mZ <- (1 / charge - eps) < diff(x$mZ) & diff(x$mZ) < (1 / charge + eps)
R> idx <- which( diff.mZ )
R> op <- par(mfrow=c(5, 3))
R> rv <- lapply(idx, function(i){
+    m0 <- x$mZ[i] -1
+    m1 <- m0 + 2
+
+    try(plot.deisotoper(x=x, y=xd, xlim=c(m0,m1),
+                     ylim=range(x$intensity[(i-1):(i+2)],
+                                -xd$intensity[which(m0 < xd$mZ & xd$mZ < m1)])))
+  })
```




```
R> charge <- 2;
R> (eps <- as.numeric(
```

```
+    as.character(config$Value[which(config$Configuration == "Delta")])))

[1] 0.15

R> diff.mZ <- diff(x$mZ)
R> idx <- which( (1 / charge - eps) < diff.mZ  & diff.mZ  < (1 / charge + eps))
R> (diff.mZ[idx])

[1] 0.50024 0.48200 0.51500 0.49300 0.49500

R> op <- par(mfrow=c(2, 3))
R> rv <- lapply(idx, function(i){
+    m0 <- x$mZ[i] -1
+    m1 <- m0 + 2
+
+    try(plot.deisotoper(x=x, y=xd, xlim=c(m0,m1),
+                        ylim=range(x$intensity[(i-1):(i+2)],
+                                   -xd$intensity[which(m0 < xd$mZ & xd$mZ < m1)])))
+  })
```



4. Implementation Details

The package is interfacing the Java code by using the the **rJava** package Urbanek (2009).

5. Session information

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

- R version 3.4.3 (2017-11-30), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=en_US.UTF-8`, `LC_ADDRESS=en_US.UTF-8`, `LC_TELEPHONE=en_US.UTF-8`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=en_US.UTF-8`
- Running under: `Debian GNU/Linux 9 (stretch)`
- Matrix products: default
- BLAS: `/usr/lib/atlas-base/atlas/libblas.so.3.0`
- LAPACK: `/usr/lib/atlas-base/atlas/liblapack.so.3.0`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: deisotoper 0.0.3, rJava 0.9-8
- Loaded via a namespace (and not attached): compiler 3.4.3, tools 3.4.3

References

Eng JK, Hoopmann MR, Jahan TA, Egertson JD, Noble WS, MacCoss MJ (2015). "A deeper look into Comet–implementation and features." *J. Am. Soc. Mass Spectrom.*, **26**(11), 1865–1874.

Panse C, Grossmann J (2012). ***protViz**: Visualizing and Analyzing Mass Spectrometry Related Data in Proteomics*. R package version 0.2.48, URL https://CRAN.R-project.org/package=protViz.

Perkins DN, Pappin DJ, Creasy DM, Cottrell JS (1999). "Probability-based protein identification by searching sequence databases using mass spectrometry data." *Electrophoresis*, **20**(18), 3551–3567.

Urbanek S (2009). ***rJava**: Low-Level R to Java Interface*. R package version 0.9-9, URL https://CRAN.R-project.org/package=rJava.

Yuan Z, Shi J, Lin W, Chen B, Wu FX (2011). "Features-based deisotoping method for tandem mass spectra." *Adv Bioinformatics*, **2011**, 210805.

Affiliation:

Christian Panse & Witold E. Wolski
Functional Genomics Center Zurich, UZH|ETHZ
Winterthurerstr. 190
CH-8057, Zürich, Switzerland
Telephone: +41-44-63-53912
E-mail: {cp,wew}@fgcz.ethz.ch
URL: http://www.fgcz.ethz.ch