chem-bla-ics

Recovering full mass spectra from GC-MS data

Egon Willighagen

Published June 4, 2008

Citation

Willighagen, E. (n.d.). In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/r7fav-92885

Keywords

Metabolomics, Rstats

Copyright

Copyright © Egon Willighagen 2008. Distributed under the terms of the Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

One aspect not covered in detail by the ongoing discussion on unit testing quality control for scientific software, is detecting regressions. This is the most important reason why unit testing is superior to random testing. Putting someone behind a keyboard to tests things is nice, but this process has to be repeated, as the testing has to be repeated over and over again. Just to make sure it works for whatever new input, for whatever refactoring, for whatever new cool feature.

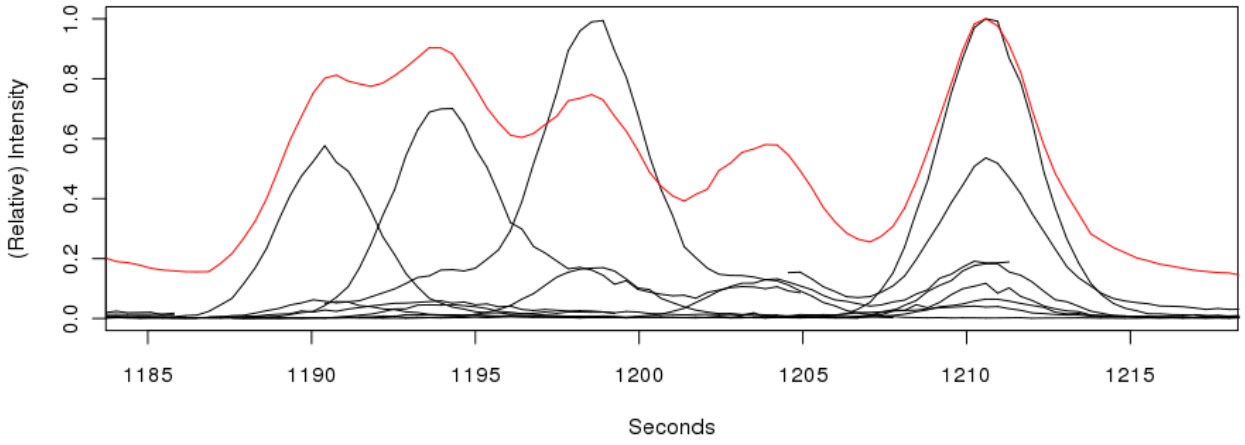
Another advantage of unit testing over random testing, is in the fact that it provides you with statistics (lies, damn lies, and statistics). These statistics do give some insight where to start looking, though if really written properly, each unit test has the ability to test a single line of functional code. That's where code coverage testing is useful, and should be part of the process too. I have no idea what commercial chemoinformatics software vendors do regarding quality control, but I assume they make heavily use of code coverage too.

SimBioSys Blog mentioned the use of annual software competitions. That's important indeed, and provides nice means to compare options, but it is not unit testing; it's a macroscopic test of functionality, and has little means to identifying underlying fails. Is a bad CASP score caused by wrong isotope masses used in the force field, or by the approach? I'm sure no one can tell.

Anyway, refactoring is a principle activity of software engineers, and unit testing manages that process in some way. Taking unit testing to the extreme, any new coding starts with writing the unit tests for the new API. Only when those are finished, the functionality is actually implemented.

Ok, now something less boring. GC-MS-based Metabolomics with metabolite identity in particular. Though I believe there are other uses too, for example, in between sample alignment, the recovery of a full mass spectrum is particularly important for metabolite identification of new, yet unknown compounds (yes, even dereplication is already non-trivial, because of the lack of free (open data preferably), machine accessible (open standards!) database of mass spectra (using different ionization methods). Look up by monoisotopic mass is possible in, for example, ChemSpider (see this blog), but look up via full spectrum is less common. The number of databases are growing, and likewise the openness and accessibility. Who know what the Netherlands Metabolomics Center's support platform will be able to offer in a year or two.

Now, unit tests could, for example, tests that some algorithm can deconvolute the following GC-MS data:



The red line is the TIC for the chromatogram, while the black lines are the extracted ion chromatograms of individual m/z (ion) peaks in the mass spectral dimension, and, only of peaks detected using XCMS using the new centWave method by Ralf. Now, the results are not perfect in this diagram, but it does seem to recognize all five eluting metabolites (that's the amount I would guess are eluting). However, I am more interested in the methods ability to recover all m/z peaks for each metabolite, to allow me to identify the structure, or at least make a best possible educated guess, or, with a bit of luck the compound is already known, I can dereplicate it against some database (Guesses differ, but, particularly in plant metabolomics, more than half of the metabolites we can now detect have a yet undetermined structure).

Now, I'm sure any method will be able to deconvolute these compounds. They are well separated, show a nice gaussian shape, and deconvolution based on just the chromatographic domain will likely already work. It starts to become more difficult for the low intensity peaks, those with low signal-to-noise ratios, or those with a different elution profile (e.g. peak tailing). Deconvolution typically requires some peak shape (commonly Gaussian or Exponential-Modified Gaussian), while experimental data typically does not have that. Jim Downing recently introduced me to the term 'Long Tail Science' (via this blog from Peter):

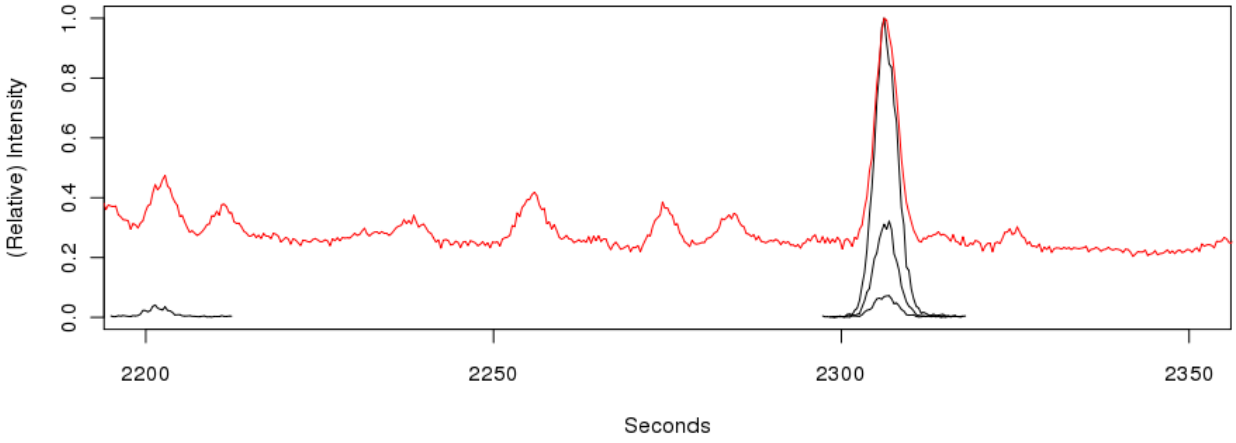
> *Jim Downing cam up with the idea of "Long Tail Science". The Long Tail is the observation that in the modern web the tail of the distribution is often more important than the few large players. Large numbers of small units is an important concept. And it's complimentary and complementary.*

This *long tail* of ion chromatograms is what I am interested. I do not care about the usual suspects, I want to learn about the 80% unknown metabolites that are found in samples. What can we learn about those? What is in the long tail of detected metabolites?

Now, I know I am not an expert in tuning `centWave` parameters, so I might as well be passing garbage in, but the method is not robust against me:

```
xr <- read(file="someClosedData.cdf")
p <- findPeaks(xr, method="centWave", ppm=150, peakwidth=c(5,25))
```

But it fails to detect some of the metabolites in the long tail:



As you can observe from the low S/N ratio on the red TIC line, you can notice that we are at low intensity metabolites. Assuming some peak shape is at such noise levels much more difficult than with better S/N ratios. The `centWave` uses a really nice non-parametric approach here… well, not entirely, otherwise it would not have failed over my parameter settings :) Steffen/Ralf, what was the DOI again?

Now, I found these missed metabolites by manual browsing the data, data exploration. SBS wrote *[t]here are four distinct type of tests: Func, Speed, Error and Robust*. I believe the above situation is really a fifth class: it's neither a true *functional test*, but it is not an true *robustness test* either. The input is valid, but of such that it moves towards invalid input. Scientific data is a continuous spectrum of input (remember the oil in, oil out). Software must be tested against such borderline data; repeatedly, over and over again, for any version, for any code change, for any platform.

Oh, and code quality has nothing to do with trust. Give me statistics (I can interpret the scope of them) over any trust assurance.