Computational Proteomics

Approaching Open Source Alternatives for MS^{E} data

Master of Science Thesis
by
Rune Schjellerup Philosof
rune@philosof.dk

Supervisor: Professor Ole Nørregaard Jensen, Ph.D.
Protein Research Group
Department of biochemistry and molecular biology
University of Southern Denmark - Odense

June 2008

Abstract

In this project, I will be investigating a mass spectrometry fragmentation method called MS^E. The particular advantages of MS^E are that it can be used for both identifying proteins and measuring those proteins' relative concentrations between two samples (also called quantitation). The two major goals of this project were: 1) Investigate the performance of a fragmentation method (MS^E) for mass spectrometers developed by Waters Corporation. 2) Incorporate support for MS^E data in an open source software project. The performance of MS^E was investigated in twofold: 1) By comparing the ability of identifying proteins in a known sample with another, more commonly used, fragmentation method (DDA), and 2) by comparing the measured protein ratios between two samples to the known concentration ratios of the two samples.

Support for MS^E data in open source software makes it is possible to experiment with this fragmentation method on instruments from other manufacturers. Additionally, it makes it possible for developers outside Waters to experiment with alternative processing algorithms and setups for MS^E data.

A lot of the free software is developed using an open source model. Open source software projects automatically enable skilled developers to inspect the code, fix errors, make enhancements, and use the software in new environments. When developing a new algorithm for analyzing some data, it makes it a lot easier, when it is possible to plug in this alternative analysis algorithms in an existing analysis platform. This way the developer can use the existing data structures for many of the subject area objects and doesn't have to implement the pre- and post-processing algorithms.

During this project, I implemented MS^E support in the open source program msInspect, then I used this program in the investigation of MS^E by comparing the performance of msInspect to that of Waters data analysis program PLGS. The expectation was not that msInspect would perform better than PLGS, but that a proof of concept support for MS^E in msInspect could be the stepping-stone for further improvements in the open source support for MS^E data. With some added effort in development of search engines, that can handle fragment spectra with multiple precursors specified, and additional improvements of the msInspect algorithms, this could become a viable platform for analyzing MS^E data. It is my hope that my efforts will aid development, when someone continues this work.

The evaluation of MS^E was overall positive, it performed slightly better than DDA, with respect to identification, and with respect to quantitation of protein ratios, it had a very high accuracy. However, it should be noted that PLGS overestimates the precision of its ratio estimate, in the calculation of its probability of up-regulation. So extra care should be taken when using these estimates.

Preface

This report, and the results in it, has been produced during my masters project in the Protein Research Group at the Department of Biochemistry and Molecular Biology at the University of Southern Denmark, 2007-2008.

The project was supervised by Professor Ole Nørregaard Jensen. I would like to thank Ole Nørregaard Jensen for his supervision of my project.

I would also like to thank Hye Ryung Jung, Msc. and Eva C. Nielsen for answering my numerous questions, running my samples and providing me with datasets, Lene Jakobsen for mixing a set of samples for me, and Thomas Aarup Hansen for proofreading parts of this report.

I thank all members of the Protein Research Group for interesting talks around the lunch table and in general for a pleasant social environment.

Most importantly, A thousand thanks to my wife Dafna and daughter Frida for their patience and love.

Rune Schjellerup Philosof

Contents

Nomenclature

MS^{E} Fragmentation method in mass spectrometry, page 19

AA Amino Acid, page 10

AMT Accurate Mass Time - msInspect, page 42

CAD Collision Activated Dissociation, page 12

CID Collision Induced Dissociation, page 12

CPAS Computational Proteomics Analysis System, page 41

DDA Fragmentation method for mass spectrometry, page 18

EMRT Exact Mass Retention Time, page 44

ESI Electrospray Ionization, page 12

ESI Electrospray Ionization, page 14

ETD Electron transfer dissociation, page 12

FTICR Fourier transform ion cyclotron resonance - mass analyzer, page 13

iTRAQ isobaric tag for relative and absolute quantitation, page 29

LC Liquid Chromatography, page 16

LCMS Liquid Chomatography Mass Spectromete, page 29

mpds MassPrep Digestion Standard - Protein mix from Waters, page 21

MS Mass spectrometry, page 11

OSS Open source software, page 38

PLGS Waters ProteinLynx Global Server - software, page 43

PTM Post Translational Modification, page 10

QTOF Quadrupole Time Of Flight - mass analyzer, page 11

rms Root mean square (of the error typically), page 24

S/N Signal to noise ratio, page 22

SILAC stable isotope labeling with amino acids in cell culture, page 28

TIC Total Ion Count/Current, page 29

TOPP The OpenMS Proteomics Pipeline, page 42

TPP Trans Proteomic Pipeline, page 40

MS^1 Survey scan, page 12

MS^2 Fragment scan, page 12

Chapter 1

Aims

In this project, I have been investigating a mass spectrometry fragmentation method called MS^E that can be used for identifying proteins and measuring those proteins' relative concentrations between two samples (also called quantitation).

The two major goals of this project were to: 1) Investigate the performance of a fragmentation method (MS^E) for mass spectrometers developed by Waters Corporation, and to 2) incorporate support for MS^E data in an open source software project. A minor goal was to compile a review of the open source software projects, that I investigated in my search of a candidate open source software project to add MS^E support to.

The performance of MS^E was investigated in twofold: 1) By comparing the ability of identifying proteins in a known sample with another, more commonly used, fragmentation method (DDA), and 2) by comparing the measured protein ratios between two samples to the known concentration ratios of the two samples.

With proof of concept support for MS^E data in open source software, it is possible to experiment with this fragmentation method on instruments from other companies. Additionally, it makes it possible for developers outside Waters to experiment with alternative processing algorithms and setups for MS^E data. It was not my expectation I would be able to make an implementation of MS^E that performs better than the commercial product by Waters Corporation.

It is my hope that this study will lay the ground further improvements in the support for MS^E data in open source software. Which would make the technology available on more platforms and to a wider audience.

Chapter 2

Introduction

First off, I am going to give an overview of what my project is about and the goals of this project. In this overview I am using some technical terms that will not be described until later in the report.

As the project title "computational proteomics" states, this project is about the various elements in proteomics that are processed computationally. This is very broad and I am indeed only looking at a few of those elements. Specifically, I am investigating a method called MS^E that can be used for identifying proteins and measuring those proteins' relative concentrations between two samples (also called quantitation).

MS^E is only available on instruments from Waters Corporation and only using their software. I will be implementing support for MS^E in open source software, both because I am an open source enthusiast, but also because it could prove interesting to test the MS^E methodology on other instruments.

The goals for this projects are to implement support for MS^E in open source software, compare the performance between MS^E and DDA with respect to identification of proteins, and investigate the performance of MS^E with respect to quantitation.

The report is structured as follows. In this chapter, I am going to explain what proteomics is and give an overview of how mass spectrometry works. The Identification chapter introduces DDA and MS^E (in terms of identification) and presents my comparison of those methods. In the Quantitation chapter, I will introduce various methods for quantitation and evaluate the performance of MS^E for quantitation of proteins. Then finally, in the Software chapter I will talk about open source software, present an evaluation of the software solutions relevant to my project, and discuss how I have implemented support for MS^E.

I have tried to avoid forward references into unread text. However, it was not possible to completely avoid bi-directional references between the chapters, but I have minimized it to only a few references to the Software chapter from the other chapters.

Before going headlong into detailed information about these topics, I will try to gently introduce what proteomics is.

2.1 Proteomics

I will start off by explaining what proteomics means and some biological significance

So what is proteomics? In biology the suffixes -omics and -omes are used extensively to denote fields of study in such a way that proteomics is the study of the proteome, which in turn refers to the totality of proteins in a cell, compartment or similar. So proteomics is the study of proteins in for instance a cell. For those not into biology, I will quickly explain what proteins are.

Our bodies are basically made of cells. The cells takes many forms, some of them are skin, muscle, or for instance nerves. There are several compartments inside a cell, which are specialized for certain tasks. Proteins are the workhorses of the cells, each performs a small task ranging from attaching a phosphor group on another protein to transporting molecules through membranes.

Proteins are composed of amino acids (AA). The AAs are connected head to tail in a chain in such a way that the primary structure of a protein is written as a sequence of AAs. A protein's sequence is encoded in the DNA. DNA consist of four nucleic acids and the combination of three consecutive nucleic acids define which of 20 AAs is placed at a position in the proteins sequence.

After the translation of DNA into a AA sequence, the AAs may be modified by for instance phosphorylations, acetylations, methylations,... Phosphorylation is the addition of a phosphate group. These post-translational modifications (PTMs) extend the range of function of a protein, such that a protein may have a different function depending on it's PTM.

Figure 2.2 shows a peptide consisting of four AAs connected in a chain. Only the backbone is shown, $R_1 - R_4$ are placeholders for the varying part that distinguishes AAs from each other. The bond between two AAs is called a peptide bond, and small molecules consisting of AAs are called peptides. Larger structures with functions are called proteins. When proteins are broken apart (digested) by trypsin, as is usually done in proteomics studies, the fragments are called tryptic peptides.

In order to figure out how the proteins work together, cells are typically treated in some way while samples of them are being taken at regular intervals. These samples are then purified, so that only the proteins remain. The proteins are then identified and the change in protein content between samples gives information about which proteins are affected by the treatment. The identification of the content is normally done by shotgun proteomics.

Shotgun proteomics implies digesting a mixture of proteins with typically trypsin and analyzing the resulting tryptic peptides with mass spectrometry. Mass spectrometry is described in section 2.2. The peptides are broken apart inside the mass spectrometer and the patterns of their fragment masses are used to identify the peptides. The set of peptides identified is then used to identify which proteins were in the sample.

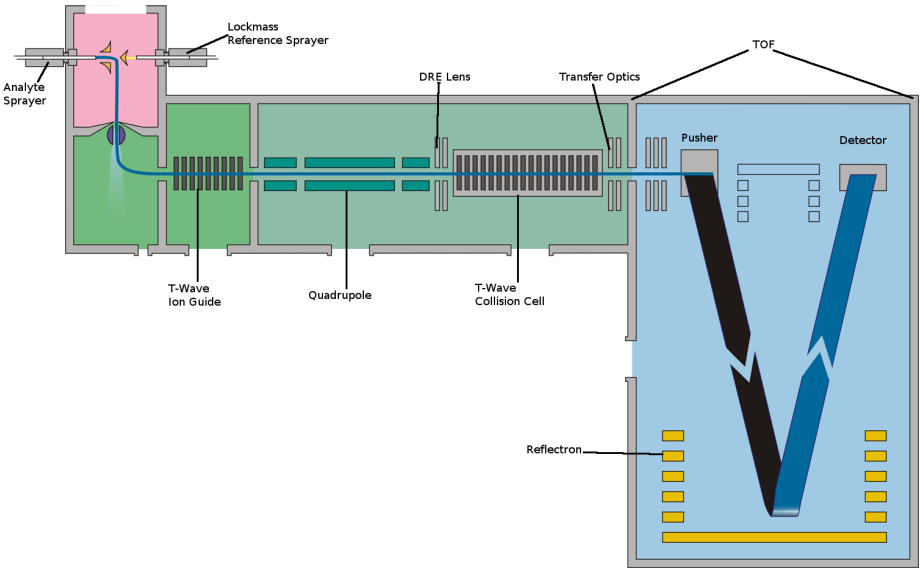


Figure 2.1: Diagram of a mass spectrometer (Waters QTOF Premier)
Adapted from the QTOF Premier Operators Manual.

2.2 Mass spectrometry (MS)

In relation to proteomics, MS can be used to identify the proteins present in a sample and with great care from the scientist it is also possible to measure the quantity of the proteins. This section will describe how MS works in general. Most information in this section (and subsections) is also described by de Hoffmann and Stroobant [21].

Mass spectrometers exist in many different types and varieties. I have been using two quadrupole time of flight (QTOF) instruments: A Waters QTOF Premier (figure 2.1) and a Waters Synapt HDMS. In section 2.2.2, I will describe how some of the instrument types work in more detail.

By using a mass spectrometer it is possible to measure the masses of components in a sample, or actually the mass to charge ratio (m/z). MS uses the fact that a charged ion in the gas phase can be attracted or repelled by electric and magnetic forces to move the ions around, filter non-interesting ions out, fragment the ions, and in the end measure their m/z.

In order to determine the mass corresponding to a peak in the spectrum it is necessary to deduce the charge state of the ions giving rise to the peak. The charge is deduced by looking at the isotopic pattern. The naturally occurring isotopes will give rise to ions that weight 1 Da more for each isotopic atom in the molecule. The ions weighing 1 Da more will appear at $\frac{1}{\text{charge}}$ higher m/z, so the charge is equal to the number of peaks in a 1 m/z wide window (not counting the peak 1 m/z higher than the first peak).

The charge is relative easy to deduce, provided that two isotopic patterns do not overlap. Estimating the charge, removing the isotopic peaks such that

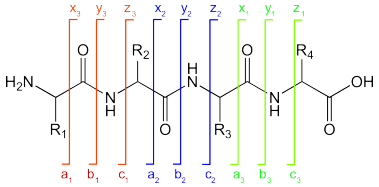


Figure 2.2: Fragmentation series of peptides
Taken from http://commons.wikimedia.org/wiki/Image:Peptide_fragmentation.svg author Kkmurray.

the ions are only represented by their mono-isotopic peak, and combining the different charge state peaks into one singly charged peak is called deisotoping or deconvolution.

As mentioned in section 2.1, ions can be fragmented in order to identify them. Fragmentation can be done by inserting a gas of nitrogen. When the ions hit the gas some of their kinetic energy is converted into internal energy which results in bond breakage and the fragmentation of the molecular ion into smaller fragments. This fragmentation method is called collision induced dissociation (CID) or collision activated dissociation (CAD). Some [43, 22] argue that the activation energy needed to generate a good set of fragment ions depends on the structure of the precursor[1], but often it is sufficient to choose the activation energy based on m/z and charge, this is how the Waters software chooses collision energy.

The fragmentation of peptides usually occur along the backbone, resulting in the ion series seen in figure 2.2. CID primarily causes y and b ion series, while for instance electron transfer dissociation (ETD) primarily results in c and z ions.

A scan with no fragmentation is called a survey scan, MS^1, or simply MS. If a mass is zoomed in on and fragmented the resulting scans are called tandem MS/MS or MS^2. If the instrument is capable of choosing a mass from the MS^2 spectrum and fragmenting that mass, the resulting scan is called MS^3, and so on. MS^E[38, 40, 39] is somewhat different, in that fragmentation is done on all masses in the survey scan instead of selecting a mass (was also called shotgun CID by Purvine et al. [34]).

2.2.1 Ion source

The sample can be either in gas phase, liquid state, or solid state. I have only worked with liquid samples that are brought to gas phase and ionized by electrospray ionization (ESI).

ESI works by having a thin needle spray a liquid at low flow. As can be seen in figure 2.3, a potential (some 3-6kV) is applied from the needle to a small slit. The optimal potential depends on the surface tension of the liquid being

[1] Dongre et al. [22] discuss SID (surface induced dissociation), but Williams et al. [43] use these results in relation to CAD without justification

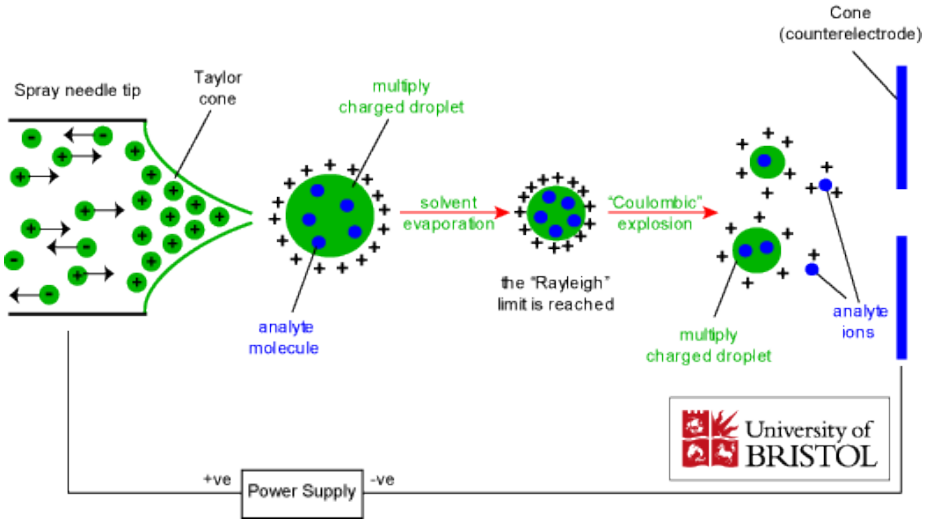


Figure 2.3: Electrospray Ionization (ESI) schematic in a mass spectrometer Figures used with permission of Dr Paul Gates, School of Chemistry, University of Bristol, United Kingdom.

sprayed. When the potential is correct, the spray forms a taylor cone at the tip of the needle, which results in small highly charged droplets.

The droplets shrink due to evaporation, which in turn causes the droplet to divide due to repelling coulombic forces ("Coulombic" explosion in figure 2.3), these smaller droplets repeat the process. In addition to dividing, the repelling forces on the droplets are be reduced by desorption of charged analyte ions.

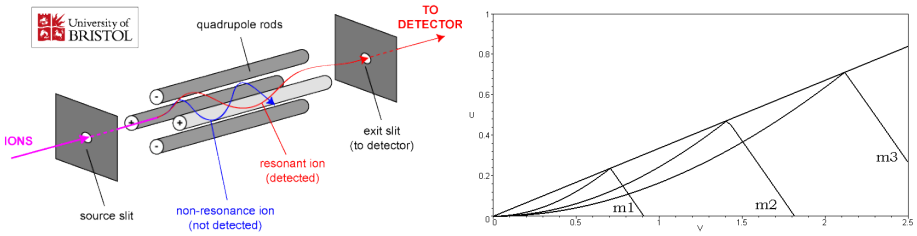
The desorption of analyte ions occur on the surface of the droplet. Since some molecules will tend to appear on the surface and others in the middle of the droplets (because of their chemical attributes), those on the surface will more easily ionize, and thus suppress the ionization of other molecules in that droplet.

In the instruments I have been working with, the needle is not positioned as in figure 2.3, but rather orthogonal to the direction the ions must move to get through the slit. The electric field generated by the potential accelerates the ions into the mass spectrometer.

It is possible to use more than one needle as in figure 2.1, thus making it possible to spray an additional sample into instrument. The contents of the second sample is known, making it possible to use it to correct m/z drift effects caused by temperature changes.

2.2.2 Analyzer

Once the ion source has created charged gas phase ions, it is time for the mass analyzer to measure their m/z. Of course many different types of mass analyzers exist and since I have only worked with QTOFs, I will only very briefly describe two alternatives: Fourier transform ion cyclotron resonance (FTICR) and ion traps.



(a) Quadrupole

(b) Quadrupole stability regions for three (unspecified) m/z values in terms of U and V.

Figure 2.4: Quadrupole schematic and a quadrupole stability region plot. The source of the right figure is questionable, but the figure looks much like those in de Hoffmann and Stroobant [21], so do not reuse that picture.
Left figure is used with permission of Dr Paul Gates, School of Chemistry, University of Bristol, United Kingdom.

Resolution defines the ability of a mass spectrometer to separate peaks. The FTICR method is a high resolution instrument, ion traps are low resolution, and TOF instruments are medium resolution.

A quadrupole can be used for filtering and fragmentation. As can be seen in figure 2.4 it consists of four rods with a potential applied to them. The potential changes polarity according to equation 2.1. When an ion enters the quadrupole it will be drawn towards the closest rod of opposite polarity, as the rods reverse polarities the ion will change direction to avoid hitting the rod.

$$\Phi_0 = +(U - V\cos\omega t) \text{ and } -\Phi_0 = -(U - V\cos\omega t) \tag{2.1}$$

In equation 2.1 ω is a constant that determines the frequency of the potential change and t is time. The terms U and V in equation 2.1 can be chosen to only let ions with a chosen m/z pass through the quadrupole. Ions not of the chosen m/z will hit the rods as seen in figure 2.4.a. Ions with m/z m1 in figure 2.4.b will only escape the quadrupole and hit the detector if U and V are set below the m1 line. As can be seen in figure 2.4.b, V determines the minimum m/z and U the maximum. By choosing U and V at the top of m/z stability region only that m/z will be let through the quadrupole. This can be used in a scanning manor, such that U and V are scanned according to the straight line in figure 2.4.b. Every scan will generate a mass spectrum.

Using three quadrupoles, the first can select a m/z to be fragmented, the second quadrupole can be used for fragmentation, and the third can then scan the m/z space to create a spectrum. When using a quadrupole for fragmentation the product ion should not be filtered away so U is set to 0. By setting U to 0 no ions of a m/z larger than the limit set by V is filtered away, instead all of the ions will be focused to a stable trajectory towards the detector. This focusing is important because the ions collide with the gas and these collisions would deflect the ions.

In a QTOF Premier and a Synapt HDMS, quadrupoles are only used for filtering. Fragmentation is done in a T-Wave, which serves as the collision cell guiding

the ions to the mass analyzer. The analyzer is an orthogonal acceleration TOF (oa-TOF), meaning the ions are pushed in a direction orthogonal to the one they entered the TOF as can be seen in figure 2.1. A TOF analyzer consists of (see also figure 2.1): 1) The pusher, where the ions are accelerated, 2) a field free region, where the ions are separated according to their velocities, 3) a reflectron, correcting for differences in velocities between ions of the same m/z, and 4) a detector.

The pusher is activated in pulses, so that the time between pushing and arriving at the detector can be measured. The time of flight can be converted to m/z using equation 2.2, where t is the time of flight, d the flight distance, e the electron charge, and V the potential used to accelerate the ions. Each pulse will generate a mass spectrum. In reality though the data from about a second (user set) will be accumulated in a single spectrum to save disc space.

$$\frac{m}{z} = \left(\frac{t}{d}\right)^2 2Ve \tag{2.2}$$

All ions will not be equally close to the pusher, resulting in different velocities for ions of the same m/z. This is corrected by one or more reflectrons (a second reflectron can be seen in figure 2.1 between the pusher and the detector), which usually consists of a series of grids and ring electrodes. Two ions with different velocities and equal m/z will penetrate the reflectron to different depths. The slow ion will leave the reflectron first but still at a slower velocity than the fast ion. The fast ion will catch up with the slow ion at the detector. The second reflectron can be used to double the flight distance d in order to increase the resolution. When used the ion will follow a **W** like flight path instead of the **V** like path shown in figure 2.1.

The detector signals a time to digital component when an ion hits it. However, the detector has a dead time after each ion where it can not detect the ions hitting it. For high intensity ion species, this dead time causes a non-linear correlation between the number of ions and the measured intensity. In addition, because the heavier ions arrive later to the detector, these are the ones the detector will miss detection of, causing a change in the shape of the peak, making it seem lighter. This effect is described by Chernushevich et al. [18].

Ion traps are basically quadrupoles bent around themselves to make a box in which ions can be trapped. Ions of a selected m/z can be ejected from the ion trap by scanning as described with the quadrupole. It is also possible to make linear ion traps that has the shape of an ordinary quadrupole and is able to function as both an ordinary quadrupole, but is also able to trap ions and eject them in an orthogonal direction through a slit in the rods.

Ion cyclotron resonance (ICR) MS traps the ions in a cyclotron which basically makes the ions travel in a circle. The ions induce a current by their movement. The signal of this current is converted to a mass spectrum using fourier transform (FT), thus the name FTICR. The detection of ions in a FTICR is non-destructive, meaning the ions can be repeatedly detected. Resolution depends on how many times the ions are detected, which in turn depends on how long the ions are analyzed, making it possible to achieve much higher resolution

than a quadrupole or TOF. It is possible to introduce a gas into the cyclotron in order to monitor fragmentation.

FTICRs are typically hybrid instruments. While the ions are being analyzed in the cyclotron, it is possible to use a linear ion trap in front of the cyclotron, typically for MS^2 analysis. This leads to high resolution survey scans and low resolution fragment scans.

2.2.3 Liquid chromatography (LC)

Chromatography is the separation of components in a mixture. Because of ionization suppression (see section 2.2.1) in ESI it is necessary to ensure that as few peptides as possible are injected at a given time. LC can be directly coupled to a mass spectrometer using an ESI ion source. Normally reversed phase (RP) LC is used. RPLC consists of a hydrophobic column (the stationary phase), as opposed to the hydrophilic column in normal phase LC. The sample is loaded onto the column by pumping it through the column in an aqueous solution (the mobile phase). The hydrophobic parts of peptides will cause them to be bound to the column material. The peptides are then eluted from the column by gradually increasing the hydrophobicity of the mobile phase. Peptides will elute in order of increasing hydrophobicity.

I have only worked with two column systems, but it is possible to use a single column. The benefit of a two-column system is faster loading time and the possibility to wash after loading. The drawback is that some very hydrophilic peptides will fail to attach to the column and thus not be observed in the MS data. The two column system consists of a small pre-column and a larger analytical column. There are three stages in a two column LC system: First loading, then washing, and finally the peptides are eluted. During loading and washing only the pre-column is used, and everything eluting is directed to waste. When switching to the final stage, the pre-column's output is directed to the analytical column, which in turn is connected to the mass spectrometer.

Chapter 3

Identification

3.1 Introduction

The samples being analyzed in our lab are typically an extract from a cell and the goal is to identify the proteins present, their PTMs, and possibly the relative quantity of the identified proteins and PTMs. Identification of proteins can be done by peptide mass fingerprinting (PMF) or by using the masses of the fragment ions of the peptides. For PMF identifications, the theoretical peptides from all of the proteins in a protein database is calculated. In this calculation a specified digestion enzyme and a protein mass limit is taken into account. The masses of the theoretical peptides are then compared to the peptide masses observed in the MS data. A protein identification is scored according to how many of its theoretical peptides are matched to the MS data. By using a fragmentation method in the mass spectrometer, it is possible to identify the peptides by their fragment ions and thus identify the proteins.

Peptides can be identified from their fragment ions using de novo sequencing, spectrum database matching, protein database matching, or a combination. Shadforth et al. [37] has written a review of the various identification algorithms available. A spectrum database search algorithm is described by Lam et al. [26]. This approach uses previously identified high quality spectra of peptides to match against. De novo sequencing tries to assign a sequence to a spectrum without a database. This requires a complete ion series to be present in the spectrum in order to determine the complete sequence of the precursor ion. Besides requiring a very high quality spectrum, de novo sequencing is also much slower than database searching, because of the much larger search space.

The most commonly used in our lab is protein database matching, typically using the Mascot[33] search engine. This works analogously to PMF, first all of the theoretical peptide sequences, which match the precursor's mass of a fragment spectrum, are extracted from the database. Then the theoretical fragments of the peptides (according to instrument specific ion series) are compared to the masses observed in the fragment spectrum. A score is then assigned to the identification based on how many of the fragments have been matched, for

many search engines (including Mascot) this score is a probability of the match being random.

According to Perkins et al. [33]: "The Mascot code iteratively searches for the set of the most intense intense peak that yields the highest score." I interpret this as, Mascot includes lower and lower intensity peak as long as the new peaks added increase the score. Messy spectra would otherwise match anything, since they have a lot of grass (low intensity ions throughout the mass range).

In the following sections I will describe two methods (DDA and MS^E) of recording MS data. Mascot expect data that has been recorded using a common method such as DDA. In particular, it does not expect data recorded with the MS^E method, so MS^E data must be made to look as much as possible like DDA data to get good results with Mascot.

3.1.1 DDA

Data dependent acquisition (DDA) is the typical method, when trying to identify the protein contents of a sample. DDA works by selecting precursors for fragmentation depending on their intensities in the preceding survey scan. Many aspects of DDA are configured by the user.

A typical configuration would include the following settings (for the Waters QTOF Premier). Consideration for the various settings are described following the list:

- Minimum intensity 40, for a peak to get chosen for fragmentation.
- A dynamic exclusion list is used to avoid selecting the same precursors several times. The entries in the exclusion list time out after 60 seconds have passed.
- The activation energy of a fragmentation scan is set according to the m/z and charge of the selected precursor ion (higher energies for higher masses).
- Survey scan lengths of ½ to 2 second. Fragmentation scans lengths are 1 second, but can be scanned 2 times if the BPI does not falls below 5.
- A maximum of 5 precursor ions are chosen for fragmentation between each survey scan.
- Lockspray scans are recorded every minute.

Setting the minimum intensity for fragmentation too low, will cause the instrument to select too many masses, that never increases enough in intensity to give good MS^2 spectra, and additionally, while scanning those too low intensity masses, some other peptides might begin eluting, that would have been detected if the instrument was not doing MS^2 scanning. Setting it too low will cause good precursors to be passed by.

When a good fragmentation spectrum has been recorded for a peptide, further spectra of the same peptide is redundant. To ensure a minimal amount of redundant spectra, the dynamic exclusion list is used. The exclusion list increases the amount of most intense ions that are fragmented. The timeout for entries in the exclusion list should be set to the time it takes for a peptide to elute. Long enough, so that a peptide is not chosen twice, but short enough for other peptides of the same mass to be selected.

Since the activation energy depends on the $\frac{m}{z}$ and charge of the precursor, only precursors whose charge can be identified by the algorithm are selected.

If the amount of precursors chosen is too high, then some of the precursors will have eluted before it is their time to be fragmented, and this will waste valuable time in which other peptides could have been fragmented.

3.1.2 MS^E

Proteomics studies usually imply very complex samples. DDA methods have the burden of only choosing a limited amount of the many peptides eluting at a time point. MS^E analysis solves this problem by fragmenting everything observed in the survey scan[38, 40, 39]. MS^E recording switches between to scan types: "Survey" and "Fragmentation". Thus, every other scan is a survey. Of course, fragmenting a wider $\frac{m}{z}$ range introduces challenges elsewhere in the processing of this data.

To identify a parent ion, its child ions are used. However, the fragmentation scan consists of child ions from all the ions generated in the ion source. So MS^E lacks the link between parent and child ions that DDA has. We simply do not know which parent ions fragmented to which child ions. However, the parent to child ion linkage can be deduced by comparing elution profiles. Fragment ions with maximum intensity at approximately the same retention time will most likely be sibling ions. Furthermore, such a group of sibling ions will have maximum intensity around the same retention time as their parent ion. The linking of parent to children through their elution profiles can be seen in figure 3.1.

With DDA the activation energy is chosen based on the $\frac{m}{z}$ and charge of the parent ion. With MS^E it is not possible to choose an energy setting this way. Waters solve this by ramping the activation energy linearly in MS^E. When the activation energy is lower than a specific parent ion's optimal activation energy, some of those ions will escape the quadrupole without being fragmented. Which is why the intact parent ion can be seen in part D and E in figure 3.1 (the $\frac{m}{z}$ denoted M+H).

MS^E fragments a larger quantity of ions than DDA, but the quality needs to good enough for an identification to be made. The quality of the peptide identifications depends heavily on the algorithm linking parent and children ions and how well the search engine handles having fragment spectra with more than one precursor. The Mascot search engine is at a disadvantage compared to PLGS, when it comes to searching MS^E data, because Mascot does not handle well, when a fragment spectrum contains fragments from other precursors than the specified. As described in the [PLGS 2.3 manual 42]: PLGS handles this by

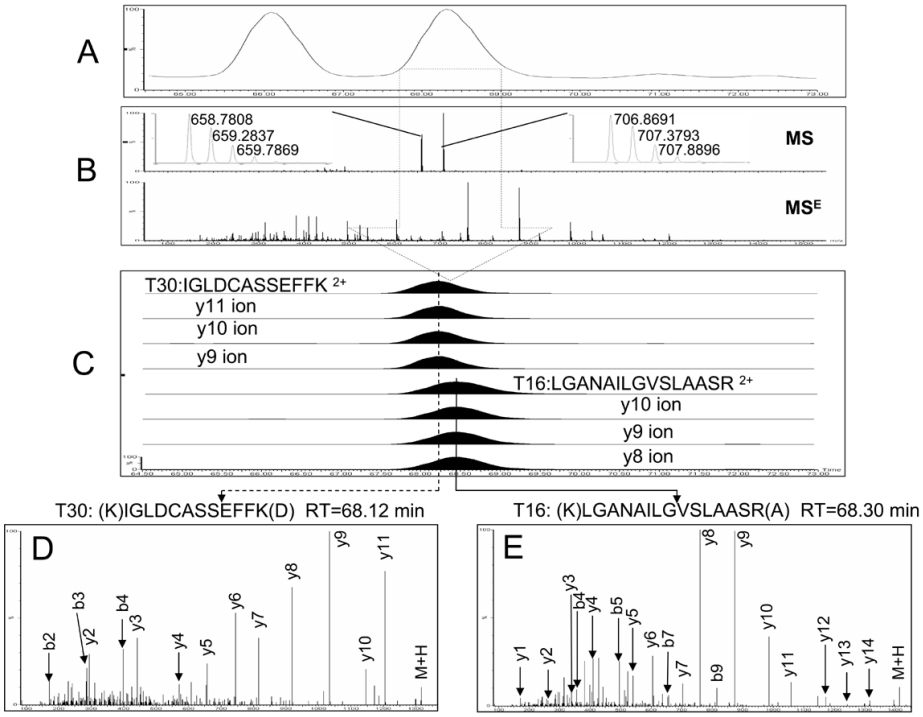


Figure 3.1: A: TIC. B: MS and MS^E spectra from the second peak in A. C: XIC of two m/z from the MS spectrum and six m/z from the MS^E spectrum. D: Spectrum containing the ions that eluted at time 68.12min. E: As D, but at time 68.30min.
Product ions in the MS^E spectrum (lower part of B) are coupled to precursor ions in the MS spectrum (upper part of B) through their elution profiles (see part C). By taking all the ions from the MS^E spectrum, with an elution profile similar to the ion with m/z 658.7808 from the MS spectrum, the identification in D can be made. Similarly for m/z 706.8691, resulting in E.
The image originates from Chakraborty et al. [17].

Proteins in mpds mix1	
ADH - P00330	Yeast Alcohol Dehydrogenase
GPB - P00489	Rabbit Glycogen Phosphorylase b
ENO - P00924	Yeast Enolase I
BSA - P02769	Bovin Serum Albumin

Table 3.1: Protein contents of the mpds mix1 sample.

iteratively removing both the parent and product features associated with the highest scoring proteins from the other lower scoring proteins. PLGS continues to do this until the false positive rate exceeds a user set threshold. Doing something similar with Mascot (without changing Mascot), would require submitting a lot of searches.

3.2 Results and discussion

The purpose of section is to evaluate the performance of MS^E with regards to peptide identifications. For this evaluation I am comparing MS^E to DDA and I am testing two processing algorithms for MS^E, the one in PLGS, and one I added to msInspect.

I have designed an experiment with samples containing varying amounts of proteins for comparing MS^E to DDA (outlined in appendix C). Unfortunately, because of instrument problems and time issues, I never got my samples run. Instead, I have mostly tested the programs using some samples that were run while testing the instruments.

For the following comparison of DDA and MS^E, I will only be using one dataset, since the other datasets I have, has problems with either the DDA or MS^E recordings. This dataset will be referred to as the Hye dataset (because she ran the samples for me). The Hye dataset consists of the four mpds proteins listed in table 3.1. The sample was run on the Synapt HDMS in both MS^E and two configurations of DDA. These configurations differed in the scan length of the survey function, which was set to $\frac{1}{2}$ s and 2 s.

The instrument was calibrated shortly prior to the runs and set to do real-time lockmass correction (unfortunately, PLGS has a bug related to real-time lockmass corrected data, that I had to work around, see section 5.2.5.6). This resulted in a dataset with a very nice mass accuracy. Since the data was already lockmass corrected, I did not have to run my own lockmass calibration on the data. This in turn removes the influence of using two different lockmass calibration algorithm, when I compare the processing algorithms of msInspect and PLGS.

In addition to the samples that were run in both DDA and MS^E, I also acquired a dataset from Waters, that was only recorded in MS^E. This dataset will be referred to as the Iain dataset and will be used for comparing different processing algorithms. This dataset consists of triplicate recordings of two samples, one containing the mpds mix1 (see table 3.1) and the other containing mpds mix1 plus an extract of E.Coli cells. The data was recalibrated using the lockmass

calibration module I added to msInspect (see section 5.2.4.2) and using the PLGS calibration option. In order to search the data, I created two databases: One containing only the four mpds proteins, and one containing the four mpds protein plus the E.Coli K12 strain proteome from swiss prot.

Before discussing the experimental results, I will compare the ratio of the signal[1] to noise (S/N) of DDA and MS^E in theory: In general, the S/N is improved by scanning the signal for a longer duration. In MS^E the duration which a parent ion is subjected to it's optimal fragmentation energy is some fraction of the duration of the fragmentation scan. However, with DDA a specific parent ion is selected fewer times than with MS^E. So, if the MS^E scans are combined this method should have the best S/N. However, my implementation in msInspect does not do this and I do not know whether PLGS does it this way. So in short: It is hard to compare the signal to noise in theory, because it will depend on the specific peptides, so let us get on with the experimental comparisons.

In the following sections, I will show the results of my comparisons of processing methods (PLGS versus msInspect), search engines (PLGS versus Mascot), and MS methods (DDA versus MS^E). Results were produced using MassLynx and msInspect for manual raw data inspection, MassWolf for converting to mzXML, PLGS and msInspect for processing, and PLGS and Mascot for searching.

The comparison are based on the quality measures that were present in both search results. Between Mascot and PLGS these measures were the amount peptides assigned to each protein, the protein sequence coverage, the rms of the peptide mass errors, and the amount of product ions matched to a protein's peptides. I included both the peptides per protein and the sequence coverage, because they don't correspond linearly to each other. First of all linearity is broken by the peptides not being equal length, but more so by the possibility of peptides with overlapping sequences (missed cleavages can cause this) and peptides that match in more than one charge state. The peptides per protein and sequence coverage are used as indicator for how well the protein was matched, while the rms and product ions are indicators for how well the peptides were matched.

3.2.1 Comparing MS^E data processed with PLGS and msInspect and searched with Mascot

I have compared PLGS processing and msInspect processing of MS^E data by searching it in Mascot. Table 3.2 shows the results from the Mascot searches. For the calculation of the results, all peptides with a score below 15 were ignored. Because of the small size of the database, the significance threshold ($p < 0.05$) was a bit below 15 for all the samples (see Perkins et al. [33] for a description of the Mascot scores and significance threshold).

As can be seen in table 3.2, it looks like the performance of msInspect is better for simple samples, but suffers heavily from the increased sample complexity, and more so than PLGS. For the PLGS processed data only 2-3 of the mpds proteins (not shown in the table) was identified, with an average of 1.25 peptides

[1] The signal being the intensities from peptide ions

MS^E samples	Processing	Score[a]	Pep/prot[b]	Coverage[c]	RMS[d]	Products[e]
Hye	msInspect	**369,58**	**19,33**	**20,69**	12,06	**114,08**
Iain	msInspect	296,58	11,08	18,12	17,47	70,25
Hye	PLGS	161,92	6,75	9,77	**3,72**	39,75
Iain	PLGS	72,5	2,67	3,62	12,62	13,33
Iain +E.Coli	PLGS	33,5	1,25	2,08	12,71	7,13
Iain +E.Coli	msInspect	-	-	-	-	-

Table 3.2: Comparison of Mascot searches on the mpds datasets using PLGS processing and msInspect processing summed up in five key measures. The measures are the mean of: a) the protein score, b) the amount of peptides per protein, c) the protein sequence coverage, d) the rms of the peptide errors, and e) the fragments of the peptides per protein. The rows are ordered from best to worst in all columns except RMS.

per mpds protein. For the msInspect processing no results at all were found. So even though msInspect seems better for simple samples, the performance of PLGS seems to drop less than that of msInspect, as the sample complexity increases. It should be said though that the tendency is on the mean response, for specific proteins the methods perform slightly different, this will be discussed further in section 3.2.1.

It would be interesting to try this comparison on medium complexity samples (medium, compared to these two) to see how complex the sample has to get for the msInspect implementation to drop below the performance of the PLGS→Mascot workflow.

3.2.2 Comparing PLGS and Mascot using MS^E data

I have compared the searching of MS^E in PLGS and Mascot both when processed with PLGS and msInspect. Comparing the search results from two different search engine is challenging, because the normal quality measure is the primary score, which is not comparable across search engines.

I used the best result from section 3.2.1 to compare the PLGS search engine and the Mascot search engine. The datasets was processed in PLGS version 2.3 using the default settings, with the addition of lock mass calibration for doubly charged species set to 785.8426 Da/e, but only for the Iain dataset. Likewise was the msInspect processed data also calibrated using the lockmass calibrator in msInspect.

Starting with the simplest sample with only the four mpds protein: As can be seen in table 3.3, PLGS outperforms Mascot in this exercise with a mean of 28.5 and 20.58 peptides per protein as opposed to 19.33 and 11.08 peptides per protein for the Hye and Iain samples respectfully. Likewise for coverage and products, indeed PLGS also matches more product ions to the peptides it matched. In the absence of a better estimator for how good the identifications of the peptides are, the amount of product ions matched for a peptide will have to do. So PLGS not only identifies more peptides per protein, but that those

MS^E samples	Workflow	Score	Pep/prot	Coverage	RMS	Products
Hye	PLGS→PLGS	3146.27	**28.5**	**43.58**	**4.72**	**409.08**
Hye	msInspect→Mascot	369.58	19.33	20.69	12.06	114.08
Iain	PLGS→PLGS	1863.22	20.58	33.15	10.4	189
Iain	msInspect→Mascot	296.58	11.08	18.12	17.47	70.25
Iain +E.Coli	PLGS→PLGS (2.2.5)	193.12	16.08	28.22	9.91	
Iain +E.Coli	PLGS→PLGS	768.65	13.75	26.12	7.15	114.33
Iain +E.Coli	PLGS→Mascot	33.5	1.25	2.08	12.71	7.13

Table 3.3: Matches by PLGS and Mascot on all three mpds samples. The table shows the mean over the three replicates and over the proteins of: The protein scores, they are not comparable across search engines. b) The peptides per protein, c) the protein sequence coverage, d) the rms of the peptide mass errors, and e) the product ions assigned to the peptides of a protein.
The +E.Coli sample measure are only for the mpds proteins identified, to make it comparable with the rest of the samples. Moreover, it should be noted that this summary only shows the general tendency of the performance of the methods, see figure 3.2 for protein specific performance.

identification are also better, in the sense that they are supported by more product ions.

Continuing with the more complex mpds plus E.Coli sample in table 3.3. PLGS (2.3) is able to identify all of the mpds proteins with a mean of 13.75 peptides per mpds protein, while Mascot was only able to identify 2-3 of the mpds proteins (not shown in the table), with an average of 1.25 peptides per mpds protein. PLGS 2.2.5 has also been included here and seems to provide a better result than PLGS 2.3. However, The PLGS 2.2.5 score is not comparable to the PLGS 2.3 score, and I was not able to extract the amount of product ions per protein for version 2.2.5, so this only leaves the rms as a measure for the quality of the peptide matches. And with regard to the rms PLGS 2.3 has a slightly lower error on its peptides. So in order to properly determine which of the PLGS versions is best, a much more thorough and time-consuming analysis is required.

Table 3.3 does not tell the entire story. If each row in table 3.3 is considered a method, then there is significant interaction between protein and method. An interaction between protein and method means: The effect of using one method over another depends on which protein is considered. See figure 3.2 for a plot of peptides per protein (Matches) versus method for each protein. More details about the interaction can be found in appendix A. However, table 3.3 is usable as an indicator for how the methods perform in general.

The results in table 3.3 have not been manually inspected, so one might think the reason PLGS performs better is because PLGS results simply contain more random hits than Mascot results. To test this, I have also searched the mpds data files on the entire swiss prot database both using PLGS and Mascot. Inspecting one of those searches, Mascot returned 14 hits, that had nothing to do with the four mpds protein, while PLGS only returned one such hit. Although granted, the falsely identified protein from Mascot are below the identity threshold and the one in the PLGS result is clearly separated from the others by a significantly lower score (which the PLGS manual states should be used as an

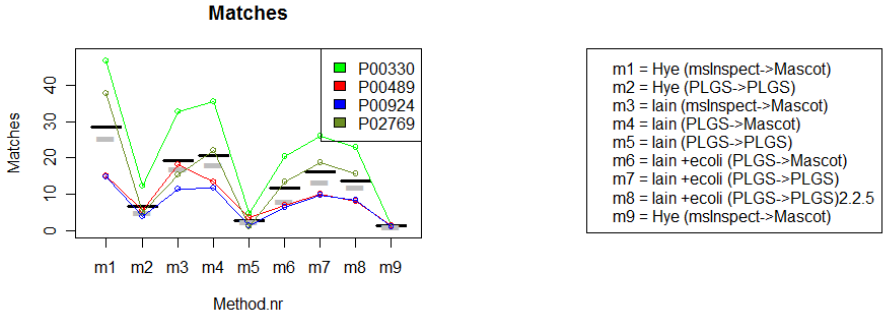


Figure 3.2: Method versus Matches specified for each protein. For the rest of the response variables, see appendix A. The plot shows interaction between methods and proteins. The most obvious example is seen for methods m3 and m4 when proteins P00489 and P02769 are compared. In this example the interaction caused the ordering of those two methods to be switched.

indicator for false identifications). See appendix B for these results.

To sum up, for MS^E data, PLGS is a better search engine than Mascot, regardless of which processing method I used to create the input for Mascot. This result is no surprise, since the search algorithm in PLGS is made specifically for this kind of data, whereas Mascot expects DDA data.

3.2.3 Comparing MS^E and DDA

For comparing MS^E and DDA, I will be using the Hye dataset, as this is the only dataset I have with comparable DDA and MS^E runs. As described in section 3.2 on page 21, the sample was recorded in triplicates both in MS^E and in two configurations of DDA. The only difference between the two DDA configurations is the survey scan length being $\frac{1}{2}$ s and 2 s. The MS^E runs also used 2 s for each scan (except lockmass scan which were 1 s).

Even though all of the results are compared in PLGS, I cannot use the primary score for comparing the DDA and MS^E runs, because PLGS uses different scores for the two methods. I even have to do without the products per protein, because PLGS does not supply this for DDA data. This means that there are no quality measures for how well the peptide were matched.

A comparison of PLGS and Mascot with regards to the DDA data can be seen in table 3.4. Mascot detects more peptides per protein than PLGS. However, with no indication from PLGS about the amount of products per peptide, this might be because PLGS uses stricter rules than Mascot for the peptides it includes. The parameters for the search engines are quite alike, but one option (validation) sticks out as being quite different. With validation turned on PLGS only includes peptides with minimum three consecutive product ions from an ion series. Executing such a filter on the Mascot results would probably reduce the amount of peptides per protein (and increase the amount of products / peptide). Turning the PLGS validation off, is not a realistic option, because

Run	Search engine	Score	Pep/prot	Coverage	RMS	Products
DDA $\frac{1}{2}$s	Mascot	750.17	**23.92**	**34.93**	3.77	**146.42**
DDA 2s	Mascot	622.83	19.75	30.98	**3.43**	120.5
DDA $\frac{1}{2}$s	PLGS merged	1.39	18	29.87	4.47	
DDA 2s	PLGS merged	1.39	16	27.14	3.81	
DDA $\frac{1}{2}$s	PLGS	1.39	15.67	26.11	4.2	
DDA 2s	PLGS	1.39	13	23.22	3.71	

Table 3.4: Comparison of PLGS and Mascot search engines on two sets of DDA data from the Hye dataset. PLGS was used for processing the input for both search engines. The measures are the means over triplicate runs and over the four proteins of these measures: Protein score, peptides per protein, sequence coverage, rms of the peptide errors, and products ions matched to the peptides of the protein. PLGS merged indicates that I used the merging feature in PLGS, with which it merges a set of search results into one result.
The rows has been ordered according to how well the methods performed (not considering RMS), and the best scores marked by bold font. The protein scores of Mascot and PLGS are not comparable.

each peptide then requires manual validation. Furthermore, it would not help in regards to comparing the results.

For the comparison in table 3.5, I only used the top scoring methods from table 3.4. With a slight increase in peptides per protein, but a substantial increase in products per peptide, MS^E comes out at the top, at least for the PLGS→PLGS workflow. Using msInspect→Mascot for the MS^E run is slightly worse than both of the DDA runs. It is not really possible to get any strong conclusions from this DDA versus MS^E result, because I only have this one observation, making it impossible to calculate the variation of the measures.

With this simple four protein mix comparison, MS^E wins by a small margin. However, it is expected that the numbers will be quite perturbed by increasing the complexity of the sample as was also seen in section 3.2.1 on page 22. Unfortunately, I did not have access to comparable MS^E and DDA runs of more complex samples with known proteins spiked in.

3.3 Sub-conclusion

The results in this chapter comprise of two comparisons: One of MS^E versus DDA and one of msInspect versus PLGS. The comparison of msInspect and PLGS processing was based on two simple samples containing four known proteins and one complex sample consisting of an E.Coli extract with the four known protein spiked in. It came as no surprise that PLGS→PLGS outperformed msInspect→Mascot in all samples. For the Hye dataset the results were a mean of 28.5 and 19.33 peptides per protein, for the PLGS and msInspect workflows respectively, adding up to mean sequence coverages of 43.58 and 20.69 respectively.

It should be noted, that I did not expect to have time to create a competing product during this project, especially because none of the search engines available

Method	Workflow	Score	Pep/prot	Coverage	RMS	Products
MS^E	PLGS→PLGS	2986.44	**25.5**	**41.81**	4.64	**409.08**
DDA $\frac{1}{2}$s	PLGS→Mascot	750.17	23.92	34.93	3.77	146.42
DDA 2s	PLGS→Mascot	622.83	19.75	30.98	**3.43**	120.5
MS^E	msInspect→Mascot	369,58	19,33	20,69	12,06	114,08

Table 3.5: Comparison of MS^E and DDA identification in Mascot and PLGS using the Hye dataset. I have only included the best scoring DDA methods from table 3.4. The measures are the means over triplicate runs and over the four proteins of these measures: Protein score, peptides per protein, sequence coverage, rms of the peptide errors, and products ions matched to the peptides of the protein.
The rows has been ordered according to how well the methods performed (not considering RMS), and the best scores marked by bold font. The protein scores are not comparable across search engines.

to me was prepared for MS^E data. However, the msInspect→Mascot workflow *was* able to identify all the proteins in the low complexity sample. Actually, for the low complexity samples, msInspect→Mascot outperformed PLGS→Mascot, although, not for the high complexity sample, for which msInspect→Mascot was not able to identify a single peptide. With the complex sample, PLGS was able to identify about half as many of the peptides from the four proteins as it had with the simple sample, and with a slightly lower amount of product matches per peptide.

With some added effort in development of search engines, that can handle fragment spectra with multiple precursors specified, and additional improvements of the msInspect algorithms, this could become a viable platform for analyzing MS^E data.

The comparison of DDA versus MS^E, was based on a sample consisting of four proteins. For this simple sample, MS^E (PLGS→PLGS) outperformed DDA (PLGS→Mascot) with 6.6% more peptides per protein, 19.7% more sequence coverage, and 9.9 more products per peptide corresponding to an increase of 162.1%. Only rms was slightly worse, it changed from 3.43 to 4.64 (↑35.2%). These are some very simple samples and it should be investigated how the results are affected, when the sample complexity is increased.

Chapter 4

Quantitation

4.1 Introduction

Ion intensities correlate with the sample concentration of the protein from which they originate. This makes it possible to use MS for relative concentration comparisons[31]. Absolute quantitation is also possible, but requires additional runs in which the concentration is varied (as described by Gröpl et al. [24], Silva et al. [40]).

Peptides are mostly quantified from their MS^1 elution profiles except in the case of iTRAQ[31, 35]. The intensity used can be the elution maximum intensity, the area spanned by the elution profile, the area spanned by a model fitted to the elution profile, or something similar. The simplest intensity measure is naturally the maximum intensity. Measuring the area is far more difficult, because it can be difficult to identify where the peak starts and ends (due to overlapping peaks and noise).

4.1.1 Labeled

I will briefly describe two labeled quantitation approaches in this section. First "stable isotope labeling with amino acids in cell culture" (SILAC)[32] then "isobaric tag for relative and absolute quantitation" (iTRAQ)[35].

With SILAC the cells are grown in media containing modified amino acids. For each state that is being compared, a separate cell culture must be grown, one set of cells will be grown in normal media and the other sets in media modified in some way, for instance in a medium where all of the arginine amino acids contain only the ^{13}C isotope making it 6 Da heavier. When the cells have grown for days they will have incorporated the modified amino acid in every protein. Samples are then extracted from the different states and mixed to make a single sample. When the sample is run on a mass spectrometer, every peptide will be uniquely represented in the survey scan for each state, with a mass distance defined by the label used. So in a sample with two states labeled with normal and ^{13}C arginine, every peptide will occur twice with 6 Da between the ions.

With iTRAQ the labeling occurs at the peptide level after the proteins have been extracted and digested. The peptide mixtures from the different states are treated such that the peptides from each state has a different label molecule covalently attached. After this labeling, the mixtures are combined to single sample as in SILAC. The label molecules consists of two subunits, a reporter unit and a balance unit. The label molecules used in iTRAQ has the masses 114-117. When the peptides are fragmented the reporter units will give rise to reporter ions that are used for quantitation. The balancing unit is included for the intact peptides to have equal mass so that the peptides from different states are fragmented in the same fragmentation scan. Since the quantitation is done in the fragmentation scan, only peptides that are selected for fragmentation can be quantified.

4.1.2 Label-free

In section 4.1.1 I described how different states can be quantitatively compared by labeling the samples, mixing them, and then analyzing them together in a single run. Another way to quantitatively compare states is by analyzing the states in separate runs without labels[40, 39]. Obviously this is an easier setup, but it at comes at a cost. Special care has to be taken to make the conditions for the runs as equal as possible, and the irremovable variation between the runs must be identified and neutralized. Variations between runs are often removed by normalizing the intensities with standards that are known to have equal concentration in the runs, or by other normalization procedures such as picking proteins that vary the least in intensities among the samples.

Several ways to label-free elucidate the relative concentrations of samples have been described in the literature, some less complex than others. Starting with the simple approaches, we have spectral counting and a variant (spectral TIC) using the TIC (total ion current) of the MS^2 scan as described by Asara et al. [15]. These simply use either the amount of MS^2 spectra assigned to a protein or the average TIC of the MS^2 spectra assigned to a protein. More complex methods require detection of features in the 3D landscape that LCMS is. Detected features from different runs needs to be matched based on their time and m/z (and possibly assigned peptide identification) in order to be compared. This approach has the advantage of not requiring a the features to be identified beforehand, but instead this approach can be used for selecting interesting features that are then identified by a targeted MS run.

4.1.3 MS^E

With label-free quantitation the survey scans are used for measuring the intensities of the features. DDA (from a QTOF) is not suited for quantitation using these methods, since survey scans are scarce, and some algorithms will not work when the scans are not spaced equally. Additionally DDA (from a QTOF) is inappropriate, because the precursors that are selected will vary between the runs, and thus the amount of, and distance between survey scans will be different for the runs. The optimal recording method for quantitation methods is MS only (only recording survey scans), however, when using only survey scans it is

necessary to run a separate set of runs for identifying the peptides, and then subsequently match the identifications and quantitations.

With DDA on a hybrid instrument like a FTICR that has two mass analyzers, the survey scans are regular and might prove useful for label-free quantitation. For these instrument the survey scans would reach much higher resolutions, which should make it easier to isolate and quantitate the peptides. However, the fragment scans are done in a low resolution analyzer.

As mentioned in section 3.1.2, MS^E switches between to scan types: "Survey" and "Fragmentation". Thus, every other scan is a survey. The issues mentioned above are taken care of with MS^E: The survey scans are equidistantly spaced, the amount of survey scans are reasonable, and identification (as described in section 3.1.2) is also possible in the same run[38, 40, 39]. Compared to only survey scans the amount of survey scans are halved, but it is much better than DDA on a QTOF for label-free quantitation.

4.2 Results and discussion

Protein	Mix 1	Mix 2	Log ratio
ADH[1]	1	1	0
GPB[2]	1	0.5	-0,69
ENO[3]	1	2	0,69
BSA[4]	1	8	2,08

Figure 4.1: Mpds mix 1 and 2 from Waters

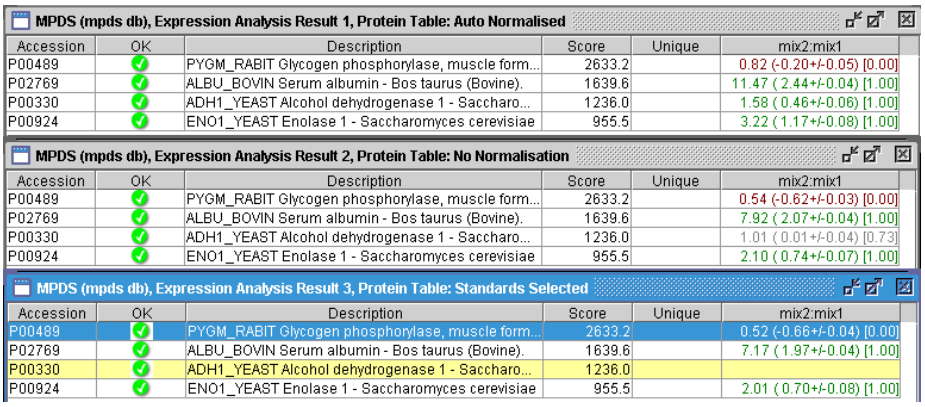
To evaluate the MS^E label-free quantitation performance, I designed two mixtures with 6 proteins. In the first mixture all proteins had equal concentrations. In the second mixture some proteins had different concentrations, see appendix C for further details on this. Unfortunately, because of instruments problems and time issues, I never got any of my homemade samples run. Instead of recording the data in our lab, I acquired samples from Waters called mpds. Mpds contains digests of 4 proteins in two mixtures as described in figure 4.1.

The mpds dataset I received from Waters was recorded on a QTOF Premier. In addition to triplicates of mix 1 and 2, the dataset also included triplicates of the two mixtures with E.Coli proteome background. The samples with E.Coli background will make it possible to evaluate the performance of MS^E with simple and complex samples.

I processed the datasets in PLGS and msInspect. Only PLGS will be able to compute protein ratios for the mpds +E.Coli dataset, because quantitation on a protein level is inherently dependent on being able to identify the proteins, and as was seen in section 3.2.1 on page 22, using msInspect→Mascot processing did not give any identified proteins. My expectation is that PLGS performs best of the tools, because it is a commercial product designed with this kind of dataset in mind.

4.2.1 PLGS

Looking at the protein tables in figure 4.2 it is obvious that, for some settings, PLGS is able to extract the correct protein ratios (see figure 4.1) from the



MPDS (mpds db), Expression Analysis Result 1, Protein Table: Auto Normalised

Accession	OK	Description	Score	Unique	mix2:mix1
P00489	✓	PYGM_RABIT Glycogen phosphorylase, muscle form...	2633.2		0.82 (-0.20+/-0.05) [0.00]
P02769	✓	ALBU_BOVIN Serum albumin - Bos taurus (Bovine).	1639.6		11.47 (2.44+/-0.04) [1.00]
P00330	✓	ADH1_YEAST Alcohol dehydrogenase 1 - Saccharo...	1236.0		1.58 (0.46+/-0.06) [1.00]
P00924	✓	ENO1_YEAST Enolase 1 - Saccharomyces cerevisiae	955.5		3.22 (1.17+/-0.08) [1.00]

MPDS (mpds db), Expression Analysis Result 2, Protein Table: No Normalisation

Accession	OK	Description	Score	Unique	mix2:mix1
P00489	✓	PYGM_RABIT Glycogen phosphorylase, muscle form...	2633.2		0.54 (-0.62+/-0.03) [0.00]
P02769	✓	ALBU_BOVIN Serum albumin - Bos taurus (Bovine).	1639.6		7.92 (2.07+/-0.04) [1.00]
P00330	✓	ADH1_YEAST Alcohol dehydrogenase 1 - Saccharo...	1236.0		1.01 (0.01+/-0.04) [0.73]
P00924	✓	ENO1_YEAST Enolase 1 - Saccharomyces cerevisiae	955.5		2.10 (0.74+/-0.07) [1.00]

MPDS (mpds db), Expression Analysis Result 3, Protein Table: Standards Selected

Accession	OK	Description	Score	Unique	mix2:mix1
P00489	✓	PYGM_RABIT Glycogen phosphorylase, muscle form...	2633.2		0.52 (-0.66+/-0.04) [0.00]
P02769	✓	ALBU_BOVIN Serum albumin - Bos taurus (Bovine).	1639.6		7.17 (1.97+/-0.04) [1.00]
P00330	✓	ADH1_YEAST Alcohol dehydrogenase 1 - Saccharo...	1236.0		
P00924	✓	ENO1_YEAST Enolase 1 - Saccharomyces cerevisiae	955.5		2.01 (0.70+/-0.08) [1.00]

Figure 4.2: Protein table from an expression analysis on the Waters mpds dataset in PLGS. The column named mix2:mix1 shows the "ratios (log ratio ±standard deviation of the log) [probability of up-regulation]. For the auto normalized data, none of the proteins' ratios are within one standard deviance of the correct ratios listed in figure 4.1. For the results using no normalization, only `ALBU_BOVIN` and `ENO1_YEAST` are within one standard deviance of the correct ratio, and for the results using `ADH1_YEAST` as standard only `ALBU_BOVIN` is more than one standard deviance from the correct ratio. The picture is a screen shot from PLGS.

dataset without E.Coli. However, when using the Auto Normalization option in PLGS, the estimated ratios are quite off. I could not find any information about how the auto normalization in PLGS works, but if it works somewhat like the auto normalization routine described by Silva et al. [39], then it picks out those EMRTs[5] whose intensities varies the least amongst all the runs and uses those for normalizing. Apparently (if this is how PLGS does it) it picked the wrong EMRTs for normalization, which is also to be expected for this kind of sample according to Silva et al. [39]. Silva states that an auto normalization routine needs a dataset in which most of the content does not change between the conditions, such as the mpds +E.Coli dataset, in order to work properly.

This four protein sample is rather simple and can as such only be used for proof of concept. The more complex sample, where E.Coli has been spiked in, is much more interesting. The four mpds protein still have the same relative ratios, but both samples have an equal amount of E.Coli proteins that may disrupt the quantitation by reducing the amount of identified peptides and by inducing false identifications. The intensities of peptides, that have wrongfully been identified as mpds peptides, will influence the protein quantitation in the wrong direction. Figure 4.3 shows the results from the mpds + E.Coli dataset. Again as with the simple sample, the automatic normalization procedure makes the worst results. For the results with no normalization, none of the protein contains the correct ratio within one standard deviance, however they are quite close. Only in the results, for which I chose `ADH` as normalization standard, does one of the proteins (`ENO1`) contain the correct ratio within one standard deviance.

[5]Exact Mass Retention Time. As described in section 5.1.1.

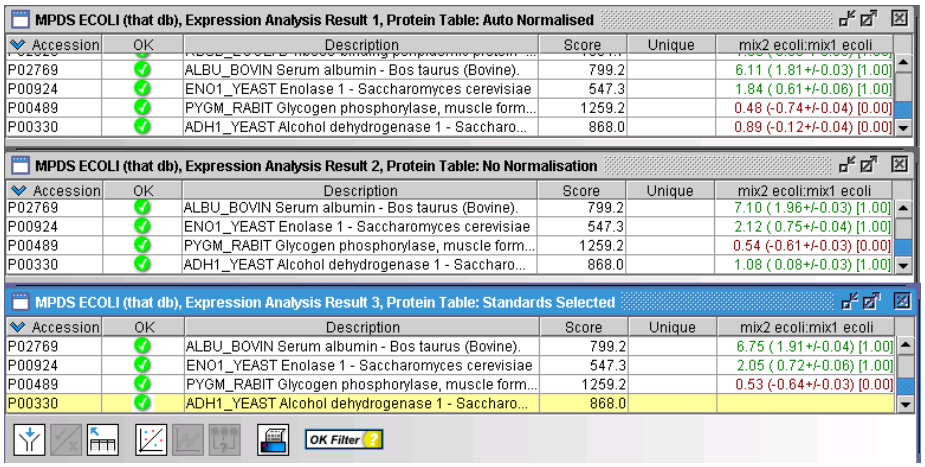


Figure 4.3: Protein table from an expression analysis on the Waters mpds + E.Coli dataset in PLGS. Only the four mpds proteins are shown.. Only one protein in all three results contains the correct ratio within one standard deviance, this is `ENO1_YEAST` using `ADH1_YEAST` for normalization. The picture is a screen shot from PLGS.

I have used a normalized rms of the ratio errors as an quantitative score of the methods. Instead taking the rms of $\text{error}_i = \text{ratio}_i - \text{correct}_i$ for all proteins i, I divide error_i by the correct ratio. By normalizing the ratio errors this way, I avoid having proteins with high ratios (like `ALBU_BOVIN`) dominate the score, because of its high variation. Using this measure it is also possible to compare PLGS and msInspect results.

$$\text{RMS} = \sqrt{\frac{\sum_{i=1}^{n}\left(\frac{\text{calc\_ratio}_i}{\text{correct\_ratio}_i} - 1\right)^2}{n}}, \qquad n = \#proteins \tag{4.1}$$

The table below shows the scores, as calculated by the above formula, for the three PLGS normalization routines on both samples.

	Auto	No normalization	Normalization using standard
Mpds	0.5710	**0.0477**	0.0643
Mpds +E.Coli	0.1380	**0.0852**	0.0977
Excluding `ALBU_BOVIN`			
Mpds	0.6100	0.0548	**0.0285**
Mpds +E.Coli	0.0819	0.0739	**0.0460**

It is expected that using the extra information, about which protein has a constant ratio, would give the best results. However, with these samples the error on `ALBU_BOVIN` becomes much larger when using `ADH` for normalizing. The normalization improves the accuracy of the other two proteins, but not enough to get a better score than using no normalization.

The ratios of false identifications will (on average) draw the ratio of the afflicted protein closer the common ratio level of the sample, because the true peptide will have come from one of the other proteins in the sample. This causes proteins, that are present in ratios significantly different from the other proteins in the sample, to be much more influenced by false identifications, than proteins with common ratios are.

Additional effects that cause a protein to be hard to quantitate are: Some peptide sequences are more easily assigned to peptides from other proteins (or just noise) than others. Peptides that gives rise to very intense signals, risk getting outside the range where the signal correlates linearly with sample concentration. Waters has documented[20] a linear correlation between concentration and signal for

As expected, the more complex sample gives rise to less accurate quantitations. It is especially `ALBU_BOVIN` that is hit, which makes perfect sense, given that the ratios of all the E.Coli proteins are 1. However, for the auto normalization the added complexity increased the accuracy, which must be because the algorithm has an easier time guessing which proteins have ratio 1, with all of the E.Coli proteins to pick from. Nevertheless it still makes the numbers less accurate compared to using no normalization.

4.2.2 msInspect

Quantitation in msInspect turned out to be quite less user friendly than in PLGS. Especially, because I wanted to use the identifications from a Mascot search. Also, the identification to feature assigning in msInspect has been designed for separate LCMS and MS^2 runs. Since my identifications come from the same run as I am using for quantitation, they can be matched unambiguously, without the mass and time tolerances that are necessary when using separate runs for the procedure. msInspect contained some support for this kind of data, but I needed to change some of the code to make it work properly. I was not able to do any quantitation analysis on the mpds+E.Coli dataset, because the msInspect processing did not provide any identifications in the Mascot search (see section 3.2.1 on page 22).

The output available from msInspect is intensities on the peptide level, I found no code for propagating those intensities to the protein level. So in order to analyze protein ratios, I made msInspect output the intensities of the identified features in a format easily loaded into R.

There are many ways to calculate the protein ratios. The common approach would be calculating the peptide ratios using the mean and then taking the mean of the peptide ratios. I thought the median would be a better choice, because it should be less influenced by outliers. To make it complete, I also included some variants, that I did not have any special expectations for (those variant are shown in appendix D). The results of using the mean and median are shown in table 4.1.

I was not able to figure out which method PLGS uses (for the mix1 intensities: 35981, 23397, 24460. And the mix2 intensities: 47036, 48382, 48025. The ratio PLGS reports is 1.99). So I was not able to test the msInspect derived feature

Protein	Mean ratio	Mean std. dev.	Median	Median IQR
All peptides (max intensity)				
ADH1_YEAST	1.120	0.291	1.030	0.1420
PYGM_RABIT	0.759	0.693	0.597	0.0919
ENO1_YEAST	2.590	2.400	1.810	0.1900
ALBU_BOVIN	9.700	14.900	6.900	3.4700
Only those peptides that are identified in all runs				
ADH1_YEAST	1.160	0.295	1.040	0.137
PYGM_RABIT	0.776	0.731	0.596	0.111
ENO1_YEAST	2.570	2.500	1.790	**0.154**
ALBU_BOVIN	**7.230**	3.140	6.900	3.130
All peptides (intensity integrated over time)				
ADH1_YEAST	1.060	**0.243**	0.991	**0.1040**
PYGM_RABIT	0.732	0.679	0.565	0.0893
ENO1_YEAST	2.500	2.230	1.890	0.3850
ALBU_BOVIN	10.100	17.200	6.960	2.2200
Only those peptides that are identified in all runs				
ADH1_YEAST	1.070	0.263	**1.000**	0.1050
PYGM_RABIT	0.749	0.716	**0.553**	**0.0884**
ENO1_YEAST	2.550	2.310	**1.910**	0.4110
ALBU_BOVIN	7.190	**2.720**	6.960	**2.1000**

Table 4.1: Results from two ways of calculating the protein ratio. For the mean column, I have calculated the peptide intensities using the mean, then taken the ratio of the two samples, and finally aggregated the peptide ratio to the proteins by taking the mean. The std. dev. column shows the standard deviation of the peptide ratios, when these have been calculated as the mean. For the median column, I have done as with the mean column, but used median instead. The IQR column shows the interquartile range of the peptide ratio, when these have been calculated as the median.
I have marked those ratios closest to the correct ratios, the lowest standard deviance, and the lowest IQR with a bold font.

intensities using the PLGS method for propagating the feature intensities to the proteins.

As expected using the median of the peptide ratios, instead of using the mean proved a major improvement. Also filtering the peptides, so that only those peptides found in all runs are included, significantly decreased the standard deviation. I was counting on this, since it makes sense that false identifications are less likely to be replicated in all runs than true identifications.

As with the PLGS methods, I have scored these methods by equation 4.1 on page 32. I have included numbers both with and without this `ALBU_BOVIN`, because of an interesting detail with one of the stranger ways of calculating the ratio.

	Mean	Median	Mean ignore peptide	Median ignore peptide
Intensity maximum	0.322	0.1290	0.202	0.169
Peptides in all runs	0.324	0.1310	0.204	0.196
Intensity over time	0.296	0.0961	0.161	0.143
As above, for those in all runs	0.291	**0.0868**	0.161	0.142
Excluding `ALBU_BOVIN`				
Intensity maximum	0.351	0.1260	0.1010	0.159
Peptides in all runs	0.370	0.1280	0.1220	0.198
Intensity over time	0.306	0.0817	**0.0753**	0.140
As above, for those in all runs	0.331	**0.0665**	0.0845	0.142

The normalized rms confirms that using the median is better. However, the above table shows a surprisingly low rms for a method, that I did not expect to give good results; the *mean ignore peptide* method. This method simply takes the mean of all the feature intensities assigned to a protein, with no regard to which peptide they originate from, and then computes the protein ratio. Somehow, this approach works better than calculating the ratios and then taking the mean ratio, especially if not considering `ALBU_BOVIN`.

With regards to intensity estimation, it seems msInspect is able to properly identify the starting and stopping points of features, since the sum of the intensities over the elution profile gives results slightly closer to the correct ratio. Additional improvements are achieved when filtering out those peptides that are not detected in all replicate runs.

According to these results, the estimator used for the ratios should be the median$\pm$IQR.

4.2.3 msInspect versus PLGS

In order to make the precision estimates comparable, I have transformed the PLGS log standard deviance into ordinary standard deviance

$$\text{std dev}_i = \frac{e^{(\text{logratio}_i+\text{logsd}_i)} - e^{(\text{logratio}_i-\text{logsd}_i)}}{2}, \quad i \in \{1,2,3,4\}$$

	msInspect		PLGS		PLGS +E.Coli	
	ratio	IQR	ratio	std dev	ratio	std dev
ADH1_YEAST	1.000	0.1050	1.01	0.0404	1.08	0.0324
PYGM_RABIT	0.553	0.0884	0.54	0.0162	0.54	0.0162
ENO1_YEAST	1.910	0.4110	2.10	0.1471	2.12	0.0848
ALBU_BOVIN	6.960	2.1000	7.92	0.3169	7.10	0.2130
RMS	**0.0868**		**0.0477**		**0.0852**	

The table above shows the comparison of PLGS versus msInspect for quantitation of the mpds proteins. With regards to the accuracy of the protein ratio estimates PLGS manages to perform better with both samples. However, the precision estimate of PLGS, the standard deviance, underestimates the error of its ratio estimate. For instance, with the mpds +E.Coli sample, PLGS estimates a ratio of 1.08, with a standard deviance of 0.03, with the correct ratio being ≈ 2.5 standard deviances away. This underestimation of the error also sneaks into the probability of up-regulation, that PLGS calculates. For the ADH1_YEAST in the mpds +E.Coli sample, PLGS calculates a probability of up-regulation of 1.00, while this protein is neither up or down regulated.

When the quantitation results need to be expressed as protein ratios, the quantitation depends heavily on the quality of the peptide identification. Enough peptides from each protein needs to be identified, to get a proper estimate of the protein ratio. This property makes PLGS a much better tool for this kind of quantitation on MS^E data than msInspect is. It also verifies the identification results in chapter 3 on page 17, since false identifications would result in bad ratio estimates.

Using the AMT module in msInspect, it should be possible to use the identifications from the samples without E.Coli to assign identifications to the features detected in the sample with E.Coli added, but I did not test this procedure. If the ratios did not need to be expressed as protein ratios, then msInspect would probably do an excellent job of estimating the intensity of the features. Sometimes the result of the quantitation analysis does not need to be matched to any identifications. For instance, different disease conditions can be grouped according to the pattern of feature intensities, and then after good markers for grouping the conditions have been found, they can be identified by targeted MS runs. For this scenario msInspect will be an excellent choice.

4.3 Sub-conclusion

It was expected for PLGS to give the best results, and that expectation was fulfilled to some degree. PLGS was indeed closest to the correct ratio with a normalized rms of 0.0477 compared to the 0.0868 that msInspect scored (both using the four protein sample). Even though the rms of PLGS is almost half that of msInspect, both are very low, indicating very high accuracy of their protein ratio estimates.

msInspect was not able to do any quantitation on the more complex samples containing an E.Coli extract with the four proteins spiked in. This is not because

msInspect was not able to find and quantitate features, but instead because the identification workflow msInspect→Mascot did not identify any proteins for this sample, as seen in section 3.2.1 on page 22.

PLGS was able to quantitate the four proteins in the complex mpds+E.Coli sample, to a high degree of accuracy, with a rms of its estimates at only 0.0852. However, the precision estimate of PLGS, the standard deviance, underestimates the error of its ratio estimate. For instance, the correct ratio of protein `ADH1_YEAST` is 1 and PLGS estimates a ratio of 1.08, with a standard deviance of 0.03, with the correct ratio being ≈ 2.5 standard deviances away. This underestimation of the error also sneaks into the probability of up-regulation, that PLGS calculates. For the `ADH1_YEAST` protein, PLGS calculates a probability of up-regulation of 1.00, while this protein is neither up or down regulated.

The accuracy with which PLGS estimates the protein ratios also indicate that the identifications by PLGS seen in chapter 3 are correct. Every false peptide identification will reduce the accuracy of the protein ratio estimate.

Chapter 5

Software

5.1 Introduction

Please note that the contents of this chapter is qualitative rather than quantitative, in contrast to chapters 3 and 4.

In this chapter, I will describe various software solutions available to analyze proteomics data and why open-source software is important, especially to me as a software developer. Furthermore, I will mention the plethora of open-source applications available, how this creates a problem with respect to the time it takes to evaluate them, and the applications I tested and found able to do label-free quantitative analysis.

A good deal of software solutions exist for analyzing proteomic data. Each of the mass spectrometer manufacturers have developed software for identification and quantitation of samples run on the respective manufacturer's instrument. Additionally, an enormous amount of free software exist.

A lot of the free software is developed using an open source model. Open source software (OSS) projects automatically enable skilled developers to inspect the code, fix errors, make enhancements, and use the software in new environments. When developing a new algorithm for analyzing some data, it makes it a lot easier, when it is possible to plug in this alternative analysis algorithms in an existing analysis platform. This way the developer can use the existing datastructures for many of the subject area objects and doesn't have to implement the pre- and post-processing algorithms. Especially the user interface development is much easier, when adding algorithms to existing software compared to making an independent program just for that algorithm.

I have reviewed a list of software in search of suitable open source projects to which I can add the feature of processing MS^E data. The criteria for such a project is that it should be well maintained, easy to use, and fairly easy to add MS^E processing to. In order to be able to test MS^E, the software also needs some label-free quantitation support. This review will be presented in section 5.2 on page 44.

The rest of the introduction will describe the processing needed for MS data, and short descriptions of the software I have been using in chapters 3 and 4.

5.1.1 Data processing

A very comprehensive review of the data processing steps needed for MS data has been done by Listgarten and Emili [28]. In contrast, I will only give a quick introduction to some of the terms, that I will using other places in the report.

It is often necessary to process the raw data before submitting an identification search, to make the peak list as close as possible to the expected theoretical peaks that the search engines look for. With Mascot[33] for instance, it might look for the mono-isotopic peaks of y- and b-ions with and without water loss. Mascot can also look for the ions in multiple charge states (given that the precursor has multiple charges). So in order to get the best results with Mascot, some data types might need deisotoping. At the very least, the data needs to be centroided before it is submitted to Mascot.

Centroided data only consist of one data point for each isotope, whereas raw data[1] consists several data points per isotope, because of both random and systematic measurement errors. MS data can be noisy, making it difficult to determine which peaks stem from noise and which from ions, and also making it difficult to precisely determine the m/z of the ions producing a specific peak.

The following uses the nomenclature of the tools OpenMS and msInspect (see sections 5.1.5 and 5.1.4).

Peak picking (or centroiding) is the art of reducing raw data to centroided data (see figure 5.1). Peak picking is typically used for DDA data, where the MS^2 data often only consist of a single spectrum per precursor ion. The goal of peak picking is to produce a spectrum that will search well, when input to a search engine such as Mascot.

At the minimum peak picking is just finding all the local maxima in a spectrum. Additional processing might involve smoothing, noise removal, fitting models of peak shapes, and much else. One such algorithm has been described by Lange et al. [27] and it is implemented in OpenMS.

When extracting information from MS^1 or MS^E data, the same peaks will appear in a time series of spectra. Because of the extra information about each peak, another name is used for this representation. The data structure used to represent the time series of isotopic peaks is called a feature (see figure 5.1). With PLGS, features are called EMRTs (Exact Mass Retention Time).

With DDA the precursor window is typically as narrow as possible to avoid fragmenting multiple peptides at the same time. With some instruments, this means that the MS^2 spectra only contain mono-isotopic peaks making it impossible to determine the charge state of a peak. As a consequence, in most file formats that search engines accept, it is not possible to annotate the charge state of the fragment ions, only the charge state of the precursor.

[1] Raw data is often called continuous or profile data.

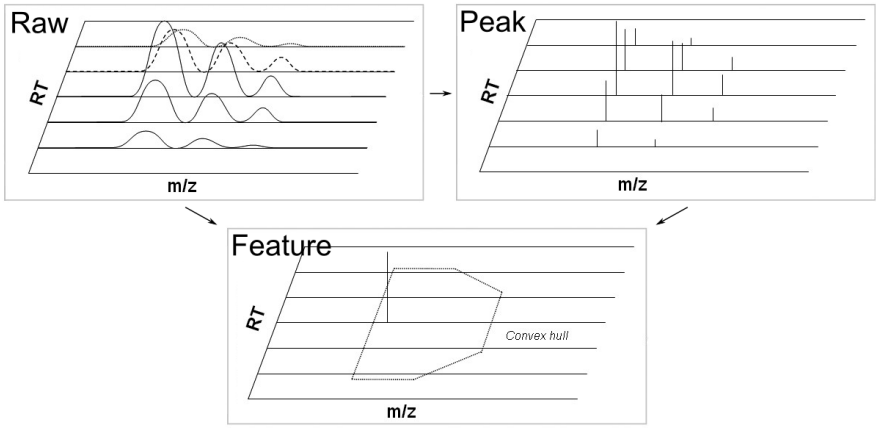


Figure 5.1: Typical data reduction for MS data. The upper left box shows smoothed raw data. The raw data is either reduced to a peak (centroided) spectrum or feature spectrum. Peak spectra consist of one data point for each isotope. While feature spectra only contain one peaks per peptide. Feature spectra may be created either directly from the raw spectra or from peak spectra.
The illustration has been adapted from Sturm et al. [41] (license: http://creativecommons.org/licenses/by/2.0/).

MS^E contains enough information to determine charge states of the fragment ions, but since it is not possible to inform the search engine of this knowledge, all peaks should instead be deisotoped, the different charge states converted to their corresponding singly charged species, and the search engine configured to only expect singly charged ions. This an example of information loss due to lacking features in search engines, another example is retention times of the precursors, which could be used for identification, as it is done in the AMT algorithm briefly described in section 5.1.4.

5.1.2 Seattle Proteome Center (SPC)

The Seattle Proteome Center (SPC)[2] maintains a substantial amount of OSS. SPC's main software is the Trans-Proteomic Pipeline (TPP)tpp [13]. TPP is a framework that interfaces most of the SPC's tools and presents them through a web server. Using TPP it is possible to do peptide validation, peptide quantitation, protein identification, and protein quantitation (labeled). Although some workflows still require usage of some command line tools.

SPC has created an open file format for raw mass spectrometry data named mzXMLmzx [9]. The format is intended as a common format for raw data from all mass spectrometers. This is important because each manufacturer of mass spectrometers have their own proprietary file format for raw data. Without a common file format, each software that needs to read raw data have to

[2]http://www.proteomecenter.org

implement readers for all the various formats instead of just the one common format.

An alternative to creating the common file format is to create a common file reader library. There are advantages to both approaches. A new format might support faster data access, smaller file sizes, and remove dependence on proprietary binary library. A file reader library avoids using time to convert the files and the space usage of having data around in both formats.

A common file reader has the huge disadvantage that, for many vendor raw formats, it is only possible to read them in Windows and many OSS projects target other operating systems like Linux and Mac OSX. With a common file format, a conversion program can convert the files to the common file format in Windows, and enable the processing programs to run in another operating system. Plus there are intentions of persuading MS manufacturers to use the common format instead of their own (or at least support it).

A part of the SPC tools is software for converting from various raw data formats to mzXML (and an upcoming mzML format[3]). The program for converting from MassLynx raw files is named Masswolf or just Wolf (the developer has not decided yet. I suggested Masswolf, when he asked for which name was best. Masswolf is easier to search for with a search engine). Masswolf is essential for working with Waters data in any of the open source tools I have investigated, since none of them read raw Waters data.

5.1.3 LabKey / CPAS

LabKey is a commercial open source project primarily developed and maintained by LabKey Corporation[4], which is situated at Fred Hutchinson Cancer Research Center (FHCRC). LabKey Corporation sells support for LabKey installations and it also receives funding from other sources. This construction ensures that the software is well maintained, which is materialized in four releases each year.

LabKey builds upon TPP and extends the capabilities in to areas like flow cytometry, assays, and observational studies. Of special interest to my work is their Computational Proteomics Analysis System (CPAS). As described in the LabKey documentationcpa [4]: “CPAS is a web-based system built on the LabKey Server for managing, analyzing, and sharing high volumes of tandem mass spectrometry data. CPAS employs open-source tools provided by the Trans Proteomic Pipeline”.

CPAS enhances the underlying tools in two ways. 1) The web-based system is a user friendly way of interfacing the underlying tools, which for the most of them only have command line UIs. 2) The results are presented to the user in customizable tables, which makes it possible to do most analysis within LabKey, or output the data in a form that is easy to work on with in for instance a spreadsheet program.

LabKey uses other tools in addition to those from TPP. For MS^1 data analysis msInspect is used.

[3] http://www.psidev.info/index.php?q=node/257
[4] http://www.labkey.com

5.1.4 msInspect

msInspectmsI [8] is an open source visualization and processing tool developed by the Computational Proteomics Laboratory (CPL) at FHCRC. It features both graphical and command line user interfaces.

msInspect has primarily been built for MS^1 data analysis as described by Bellew et al. [16]. According to mail correspondence with a msInspect developer (Damon May), they are currently working on improving the Accurate Mass Time (AMT) module in msInspect. The current AMT approach is described by May et al. [29] it consists of a database of previous MS^2 matches, these matches have had their retention times normalized. msInspect then uses this AMT database to assign identifications to MS^1 features by normalizing their retention times and searching for matches in the AMT database based on retention time and mass. This feature is alike the PEPPeR algorithm, described by Jaffe et al. [25], in some aspects[5].

In addition to the AMT module, msInspect is also able to align multiple MS^1 runs, which can be used for label-free quantitation analysis. However, after the alignment, further analysis must be done in another program. In the user guide for msInspect, it is suggested to use microarray comparison software for the final quantitative analysis, however, I imported the data into R for this part. In the near future it should be possible to analyze the label-free analysis results from msInspect in LabKey according to the LabKey development roadmap for the future[6].

I used their existing framework to create two modules for processing MS^E data. One module uses the existing feature finder in msInspect for extracting product features from the MS^E data into a featureset. The other module produces peak lists given a featureset of parent features and a featureset of the product features. See section 5.2.4 for further details.

The feature finder is described by Bellew et al. [16]. Essentially it works by: 1) Resampling the each scan into 36 bins per $\frac{m}{z}$ unit, 2) running a wavelet transform on the resampled data to sharpen the signals, 3) extracting all local maxima in the wavelet transformed spectra, 4) detecting the elution start and end of those local maxima, and finally 5) combining the maxima into a feature by comparing the maxima to the a theoretical isotopic peak distribution. The algorithm is fast enough to mark the features of a zoom within a few seconds, while inspecting the data in the GUI. Most of the speed is gained because the spectra are simplified by the rather coarse resampling.

5.1.5 OpenMS / TOPP

OpenMSope [10] is an open source development framework for mass spectrometry developed by researchers at three German Universities in Tübingen, Berlin

[5]The collaboration is also mentioned on FHCRCs homepage `http://www.fhcrc.org/science/international_biomarker/news/2007/december.html`

[6]The development roadmap `https://www.labkey.org/wiki/home/Documentation/page.view?name=roadmap` does not mention msInspect, but I presume they are going to use msInspect for the quantitation.

and Saarbrücken. TOPP (The OpenMS Proteomics Pipeline) is a set of tools based on OpenMS. TOPP contains tools for peak picking, quantitation, alignment of runs, and storage of data in a database.

I was counting on using and extending this software as I did with msInspect. Unfortunately, as it turned out I used a lot of effort, but made little progress, mostly because I could not get their feature finding algorithm to work.

I mostly tested version 1.0. However, at the end of my project (April 2008) a new version was released (version 1.1), which I did not have time to test. Version 1.1 has been improved in several aspects. It works in Windows, whereas the older version only worked in Linux. The feature finding algorithm has been improved and the parameters exposed to the user has been made more intuitive.

All tools except TOPPView are command line oriented. The TOPPView GUI has a graphical interface for some of the tools, but by design it is not capable of handling large files. According to their tutorial, you are supposed to extract a small portion of your dataset using a command line tool, then find optimal settings for the tools (Peakpicker, FeatureFinder, or what tool you want to use) using TOPPView, and finally apply the tool using the command line UI to the complete dataset.

OpenMS implements several algorithms for peak picking and feature finding. OpenMS in general and the algorithms in detail are described in the following articles: Sturm et al. [41], Schulz-Trieglaff et al. [36], Lange et al. [27], Gröpl et al. [24]. The feature finding algorithm of OpenMS is somewhat more complex than that of msInspect. Where msInspect just detects local maxima, OpenMS fits a theoretical model to both the $\frac{m}{z}$ and time dimension. This takes considerably more time than the msInspect algorithm (at least in version 1.0).

5.1.6 Waters ProteinLynx Global Server (PLGS)

PLGS is a complete analysis pipeline, from raw data processing to identification and quantitation. Three types of LCMS data is processable: DDA, MS^1 and MS^E. The identification module supports two search engines: PLGS and Mascot (it is also possible to combine the results from the two search engines). Quantitation is possible with both labeled data and label-free data (with MS^1 and MS^E). PLGS only works with Waters raw data files. However, it is possible to import peak lists (of various formats) and search these.

I have mostly been using PLGS version 2.3 and where nothing else is noted I refer to that version of PLGS. Although, I will also be mentioning version 2.2.5, because of reasons discussed in section 5.2.5.

The largest changes, from version 2.2.5 to version 2.3, have been made to the parts dealing with MS^E data, including processing parameters, search parameters, and search results. In PLGS version 2.2.5, the sets of parameters for DDA, MS, and MS^E are very alike. Except that with MS^E it is not possible to change the mass tolerances for the search engine. The search results of DDA and MS^E also different scoring mechanisms, but both include a probability score in addition to their method specific score. With PLGS version 2.3 the set of parameters for MS^E was greatly simplified and in the MS^E search results the probability score for proteins has been removed.

As mentioned above, PLGS is also able to do quantitation analysis, both labeled and label-free (requires an add-on). It is both possible to extract an EMRT (Exact Mass Retention Time) table[7] and a protein table. The EMRT table provides more control over what peptide identification each feature is assigned, but in order to get protein ratios, you have to export the EMRT table into R or a spreadsheet and calculate the protein ratio yourself. The EMRT table allows assigning identification to the EMRTs using search result from others MS runs (similar to the AMT module in msInspect). The protein table on the other hand, is the fast and easy way to get protein quantitations, but only when the search results are part of the same run, so in essence only for MS^E data.

With regard to quantitation parameters, PLGS has three normalization options: Turned off, automatic, and manual. With manual the user selects some features (for EMRT) and/or proteins (for protein table) that should be used as internal standards for normalization. The manual does not describe how the automatic normalization works.

There are some name confusions with this program, most people refer to it as PLGS. However, the user interface to PLGS is called ProteinLynx Browser. So the program the user starts is called ProteinLynx Browser, but most people call it PLGS. I presume PLGS is more popular because it is faster to say and write, so I will also be calling it PLGS in this report.

5.2 Results and discussion

As mentioned in the introduction a huge amount of free software is available for proteomics analysis. The quality of the free software varies from useless to very high quality offering optional paid support. This is a problem because it takes a lot of time to look through and test the various programs available to find those applicable to your own problems.

A substantial amount of the free software, it seems, has been created for the sole purpose of getting an article published. After the article is published, the program is either abandoned by the authors or further development is not released to the public.

There is a long way from making a program that works for the author on a specific dataset to having a high quality piece of software that can be installed and used by random users on their own computers and on different data types. The latter is not required for publishing an article, and thus the quality of a lot of the software published is low. The low quality of the software would not be a problem, if the authors were honest and tagged their programs as needing further development before widespread use. Unfortunately, this is not the case, so a lot of users use a lot of time on testing the useless programs. Or even worse, some will think a low quality program works for them and base their research on it, resulting in misleading results. However, when unmaintained software is filtered out, a lot of very useful free software is available.

[7] EMRT is the name PLGS uses for features. So an EMRT is similar to an msInspect feature and a OpenMS feature.

People should be very careful using programs that have not been updated since publication, as all software contains bugs and things to improve, so if the authors of the program are using it, then they will most likely continue to improve it for their own use. As I see it, there are three possible reasons why a program is not updated: 1) The developer has left the research group. 2) The program is not used. 3) The updates are not released to the public. All three reasons will with time make a program useless. At least outside the environment that the program was created for.

For a OSS project to be properly maintained it needs to have a sufficient developer base. There are basically two ways to establish a large enough developer base: 1) Obtain funding to hire paid software developers. 2) Attract a large enough amount of users that have some software development skills and make it easy enough for those users to contribute to the development.

5.2.1 Software review

I have been searching the literature and the web for OSS made for proteomics and specifically software that complies with my criteria: It should be well maintained, easy to use, and fairly easy to add MS^E processing to. In order to test MS^E, it also needs some support for label-free quantitation. Since MS^E support is low level processing and quantitation only makes much sense when propagated to the protein level, this requires system with a wide processing pipeline coverage.

By searching the web for proteomics tools, I found two kinds of sites that list proteomics software. The first kind of sites are general collections of proteomics software such as MS-utils.org[8], bioinformatics.ca[9], and proteomecommons.org[10]. The second kind of sites are academic groups or centers that has a list of the software that they developed, for instance NCRR[11] and SPC[12]. These sites list a large amount of software. However, none of them were good at discriminating between good and bad software.

When searching the literature for reviews of proteomics software, the results are less overwhelming and the descriptions of the software are more in-depth, making it easier to choose which software to investigate further and which to leave alone. However, few of the reviews have the same focus as I do, for instance many software reviews focus on labeled quantitation software or identification software, such as Shadforth et al. [37], while I focus on low level processing and label-free quantitation.

I would like to draw out two good reviews: 1) Codrea et al. [19] have described four tools from a users perspective, the tools were msInspect, MZmine, MSight, and MetAlign. Of those tools, only msInspect and MZmine are OSS. 2) Mueller et al. [30] have described 21 quantitation tools, of which 12 are for labeled quantitation, and nine are for label-free quantitation, of the label-free tools

[8] MS-utils.org `http://www.ms-utils.org/wiki/pmwiki.php/Main/SoftwareList`
[9] Bioinformatics.ca `http://bioinformatics.ca/links_directory/?subcategory_id=99`
[10] Proteomecommons.org `http://www.proteomecommons.org/tools.jsp`
[11] NCRR `http://ncrr.pnl.gov/software/`
[12] SPC `http://tools.proteomecenter.org`

five are OSS. Those five OSS tools for label-free quantitation are SpecArray, msInspect, TOPP, PEPPeR, and SuperHirn.

Of the OSS tools I found msInspect and OpenMS/TOPP to be most promising. Both of these tools are being actively developed and their processing pipeline coverage spans from pre-identification to post-identification[13]. I have spent much time with these projects both on familiarizing myself with their source code and on adding MS^E processing to them. Unfortunately, I did not have time to complete my MS^E implementation in OpenMS, so this has not been included in my results in chapters 3 and 4.

Table 5.1 more or less lists the programs, that I have looked at during this project. Some of the software I only briefly tested and others in detail. The table is ordered by importance to my project. The two Waters programs list highest because MassLynx is mandatory for recording MS^E data and PLGS is still necessary for getting reasonable results with MS^E data, as can be seen in chapters 3 and 4.

I recommend trying out LabKey as a data analyzing system, I think many in our lab could get more out of their analysis by using this system. However, it does need some additional development in some areas to fit neatly into our environment. At the moment, the only conversion tool automatically working in LabKey is reAdW, the converter for Thermo Xcalibur .raw files to mzXML. So converters for the other instruments in our lab needs to be added to LabKey. In addition LabKey also needs to be able to process the MS^2 spectra before sending them to a search engine, it seems the current users either process their files before uploading them to LabKey or have the instruments record in centroid mode.

OpenMS and MZmine are both promising projects for adding MS^E to. However, OpenMS needs some work on the user-friendliness side, because most of the tools require command line usage and the results files needs to be presented visually, if it was up to me, I would probably interface it to LabKey to solve those issues. MZmine needs either to extends it's pipeline coverage to include identification or preferably be able to export it's results in a format that makes it possible to easily do further analysis on them (or both of course).

Xcms had a problem with large datasets, because it load everything into memory. This is a problem, since 32 bit operating systems limit the amount of memory each process may reserve to around 3-4 GB. This users themselves can solve this obstacle by switching to a 64 bit operating system, although the best solution would of course be to change the code, to load less data into memory at the same time. I tried giving it a chance in a 64 bit linux system, but some of its prerequisites would not compile, so I never got it running in linux. Xcms is especially interesting, because it is written in R and LabKey supports easy integration of R applications in it's workflow.

5.2.2 LabKey / CPAS

As mentioned above I was quite impressed with LabKey. It has the potential of becoming a very powerful and at the same time user-friendly analysis platform.

[13] for msInspect, I included LabKey when considering this.

Name	Comments
Waters MassLynx	An absolutely necessary program, since it is the program that collects the data from the instrument. Good for visualizing Waters raw data and it has a lot of data processing algorithms, but they mainly operate on a single spectrum at a time, which is useful for checking if the instrument runs ok, but it would be nice if the spectrum processing could be executed on all spectra in the file. However, it is possible to run peak picking and calibration algorithms on the entire file (though not for MS^E data). Several extension exist that add functionality.
Waters PLGS	Still important for getting proper results with MS^E analysis, since my implementation did not perform as well. Useful for label-free quantitation using MS^E. The newest version 2.3 has more bugs than version 2.2.5 it seems. Be prepared for spending time figuring out what the error messages mean, if you are doing anything non-standard.
Masswolf[7]	Necessary in order to convert from Waters raw files to mzXML. The implementation could be much more complete, for instance information such as instrument model is hardcoded to "Q-TOF Micro". Actually the model information is not present in the Waters raw file, but Masswolf could ask the user for it.
msInspect[8]	Visualize data and perform processing on MS^1 and MS^E data. See sections 5.1.4 and 5.2.4.
Labkey (CPAS)[1]	Very active development. It is very well documented. See section 5.1.3 for description.
OpenMS/TOPP[41, 10]	Active development, new version just released. Low level processing, alignment and quantitation support. Could benefit from a more user friendly interface and support for visualization of it's results.
MZmine[2]	Old release from 2006, but a new release is scheduled for Sep. 2008. Low level processing and alignment support on MS^1 data. It needs to be able to export the processed data in mzXML format or similar.
TPP[13]	Very active development. Web-based processing pipeline for post-identification validation and labeled quantitation.
xcms[14]	Active development, but I did not get it to work with my datasets, Runs out of memory on a 32bit Windows system with large datasets, had problems installing on a 64bit Linux system.
PEPPeR[3]	Unknown last update. Runs in a web-based system called GenePattern, which I did not find easy to use. Needs better use of standard formats and visualization of results.
SuperHirn[12]	Last update in Nov. 2007 (version 0.05 beta), developed by SPC. No binaries available, so needs to be compiled by the user.
SpecArray[11]	Unmaintained (last update in May 2007), developed by SPC.
MapQuant[6]	Unmaintained (last update in 2006 and dead links on homepage), and it doesn't use standard formats.
lcms2d[23, 5]	Doesn't work on the supplied test dataset and inspecting the source code confirms that it wouldn't work on any dataset. The version released is basically a flawed version that has never worked.

Table 5.1: Overview of software for processing LCMS data. All software listed is OSS, except the two Waters program at the top. Listed in order of how promising the programs are, with respect to my criteria mentioned first thing in section 5.2.1. The programs SuperHirn and SpecArray were not tested.

In the most recent version (8.1), LabKey is not able to automatically use Waters raw files because Masswolf has not yet been included in the build. However, browsing through the source code I found that some effort has been made to support MassWolf. The required changes to make LabKey work with MassWolf is probably only pointing it to the right location and adjusting some parameters.

With the soon to come addition of label-free quantitation support (other than spectral counting), it will be a very attractive product. Unfortunately, I became interested in LabKey too late in my project to be able to use LabKey for any of my results.

5.2.3 MS^{E} in Masswolf

In order to analyze MS^{E} data with OSS, I had to fix bugs and add features to the software necessary for creating mzXML files (Masswolf) and analyzing the mzXML files (msInspect).

The first problem I ran into with respect to Masswolf development was that Masswolf is developed in Visual Studio by it's maintainers. This meant that I was unable to compile it, since I was using the free version Visual Studio Express. I wanted to make it work in Express, both so I did not have to acquire the non-free version, and to make it easier for other developers in the future to help with the development of Masswolf. Luckily, It was only necessary to replace one function call (date conversion) to another function call that used the open source library Boost.

When I started working on making Masswolf convert MS^{E} files, I found and fixed several bugs. All of my work has naturally been sent to the maintainer of MassWolf, who in return flattered me by including my name in the usage output of the program.

mzXML does not support given a scan the type `calibration` (this should be fixed in the next version of mzXML). However, Masswolf set the scan type to `calibration` anyway. Besides the fact that `calibration` scan types are not allowed, Masswolf actually gave scans two scan types, because the `calibration` was just added to the attribute list. I changed that, so scans from the reference sprayer were only given the `calibration` scan type. Furthermore, Masswolf did not know how to identify which functions were from the reference sprayer. It defined every function above the first to be a "calibration" function. I changed the logic, so now the last function (if of type "MS") is defined as `calibration` function. I also added an option to indicate that none of the functions should be defined as `calibration`.

With regards to converting MS^{E} data, I added an option to Masswolf to indicate that the data being processed is MS^{E} data. When the option is used, massWolf assumes the data consists of three functions: Survey, fragmentation, and reference. The data is saved as with DDA data, just without the precursor attribute. This means, the survey scans are saved as msLevel = 1, the following fragment scan is nested in the survey scan element with msLevel = 2, and the reference scans are saved as msLevel = 1 with scanType = calibration.

5.2.4 msInspect

For processing the MS^E data I chose msInspect. This program is primarily made for analyzing survey scan data and thus perfectly fitted for MS^E data, which is basically just two survey functions in the same run.

While implementing the MS^E support and testing the feature finding algorithm, I came around much of msInspect's source code, found bugs to fix, and located code to speed up. The bugs were mostly found in areas of the code, that the current developers do not use much, for instance the GUI (they mostly use it as a command line tool). However, one of the things I like most about msInspect is it's graphical presentation of the data and the results. While using the GUI, I ran into some issues which I fixed. For instance, I sped up the GUI in regards to drawing spectra, which was very slow when viewing the entire mass range instead of a zoom. On the usability side I added help texts, that are shown when running command line modules from the GUI. Previously, those help texts were only available when calling the modules from the command line. I have made two new command line modules. One module for MS^E, the other is for calibrating data using the calibration scans.

5.2.4.1 MS^E processing in msInspect

msInspect was only able to work on those scans with msLevel = 1 in the mzXML file, so to make msInspect able to work on MS^E data, I made it read in all the scans and added the possibility to switch to the next MS level in the GUI. Calibration data is somewhat special because it is msLevel = 1, but does not belong with the rest of the survey scans because it is a separate sample. To handle this I read in calibration data as an imaginative MS level 4. With those changes it is now possible to view all of the data in msInspect, also MS^2 scans from DDA data if need be.

The MS^E workflow, to generate peak lists for the search engines, is as follows: Apply the feature finder to MS level 1 and then to MS level 2, calibrate the results if need be (see section 5.2.4.2), deconvolute the fragment features, and finally apply the MS^E featureset combiner to generate a peak list.

In the following description of the MS^E featureset combiner, when I refer to a feature being in a scan, that feature has it's elution peak in that scan. The combiner links a parent feature to product features by assigning to the parent feature, all of the product features from the two neigboring fragment scans. Initially, I intended to use the presence of intact parent features in the fragment scans to only assign one of the two neighboring fragment scans to the parent ion. However, it turned out that a significant amount of parent ions are detected in both of the neighboring scans, meaning that one charge peaked in one scan and another charge state peaked in another scan. The following table shows the amount of parent features for which the parent feature was found in none, one, or both of the neighboring fragment scans.

Repeat	In none	In one	In both
1	2088	49	476
2	2108	49	525
3	2221	75	493

When so many parent features have two charge states features in different scans, it is reasonable that the fragment features are also spread out into both scans, so I chose to include the fragments from both neighboring scans.

5.2.4.2 Lockmass calibration using msInspect

As mentioned, SPC manages the development of Masswolf for converting Waters raw data files to mzXML/mzML files. In addition to Masswolf, SPC has also developed a program called mzXMLRecalibrator, which will recalibrate an mzXML file using the lockspray scans in it. However, mzXMLRecalibrator does this in a way not applicable to some of my files. mzXMLRecalibrator assumes the lockmass is the base peak in the lockspray scan. In my files the base peak is often very different from the lockmass, because the sample cone voltage is adjusted to ensure the intensity of the lockmass is low enough that deadtime effects are negligible.

I have implemented a method for calibrating the features found with msInspect using the lockspray function in the raw file. This enabled me to calibrate the files where half of the lockspray scans were strange (see section 5.2.7), by only using the normal looking scans.

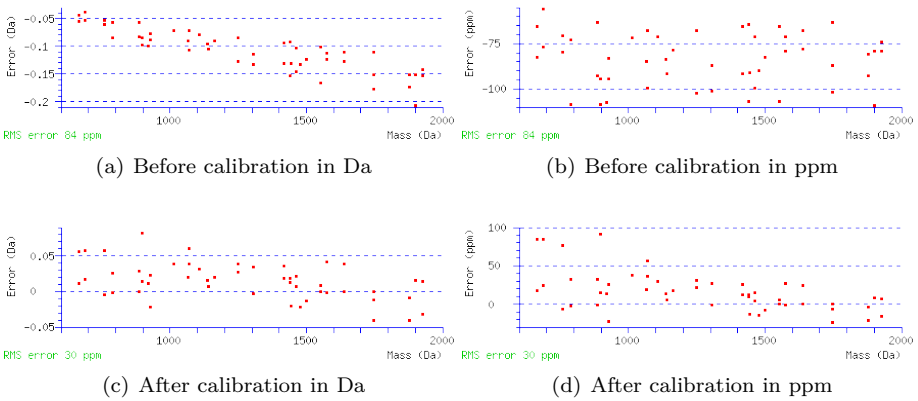
The lockmass peaks for calibration is found by looking for features in each spectrum in a m/z window around the lockmass (-1 m/z, +4 m/z). If more than one feature is found, only the one closest to the lockmass will be chosen. Using this set of lockmass features a feature file can be calibrated.

Calibration is done by using the average m/z (**a**) of the two lockmass peaks eluting before and after the feature to be calibrated. The features have their m/z corrected by the expected lockmass m/z minus **a**. Alternatively it can be chosen to have the m/z corrected by the difference in PPM. This way a 0.1 m/z difference for the lockmass 785.8426 m/z (from GluFib) which is 127.25 PPM, will result in a correction of $\approx$ 0.128 m/z on a feature at 1000 m/z. MzXMLRecalibrator uses the latter way when calibrating and this is also the default choice in my implementation.

As can be seen in figure 5.2.a the mass error does indeed increase with mass. The error measured in ppm does not increase with mass as seen in figure 5.2.b. By calibrating the data using my new module for msInspect the RMS (root mean square) is decreased from 83 ppm to 30 ppm. It is also clear from the figures 5.2.c and 5.2.d that the errors have been more or less centered around 0.

5.2.5 Waters PLGS - Problems and Pitfalls

This section contains my evaluation of PLGS. The section header is called problems and pitfalls, because I have had quite a lot of problems with this software,



(a) Before calibration in Da

(b) Before calibration in ppm

(c) After calibration in Da

(d) After calibration in ppm

Figure 5.2: Mass error plots from Mascot in both Da and ppm, before and after calibration.

that I did not expect from a commercial product, mostly with various types of data that PLGS did not support. In themselves the unsupported data types are not problematic, the problem is that PLGS does not display a usable error message, telling the user what the problem is. Due to the lack of proper error messages, I have spent a lot of time on figuring out what went wrong, time that would not have been wasted, had I been given a descriptive error message. As a result, most of this section will deal with the problems I encountered.

In summary, my general impression, from my experiences with PLGS versions 2.2.5 and 2.3, is that they have only focused on improving the MS^E pipeline and that their MS^E processing is more new than robust (small changes in the quality/type of data will stop the program). However, PLGS does work very well for some tasks. If you have good quality MS^E data or DDA data and you want to do identification and/or label-free (with MS^E data) quantitation or iTRAQ (with DDA data) quantitation, then PLGS should be able to provide nice results, as is seen in chapters 3 and 4.

I have run into several problems while trying to use PLGS. Unfortunately for the hapless user, PLGS usually does not provide any useful information in it's error messages, sometimes it does not even indicate that an error has occurred (or you have to notice the text *Error* in lower right corner before it disappears after 10 sec. or so). In addition, being able to run a dataset at one time, does not necessarily imply that it will work again at a later time.

If you for some reason have to work with PLGS and something seems not to work, you should look for the directory "c:\plgs2.3\log\" in which you will find files, that may contain an error message (typically in the form of a java stack trace and error message) that may lead you to a solution/workaround. Unfortunately, I had to work with PLGS in order to process my MS^E data, so I have used several weeks trying to make it do my bidding. Contacting Waters' application support is the way to go.

James Langridge from Waters strongly encouraged that we use the latest version of PLGS, namely version 2.3, because of new algorithms, that should give better

results. In general I agree that the newest version of a piece of software should be used, at least when the new features extend into backend algorithms. It is obvious that a lot of work has been put into new processing and searching algorithms for MS^E data, and in general they seem to give better results. But while the focus has been on the performance of the new backend algorithms, it seems they have not had time to properly quality assure the software, and many slightly suboptimal datasets will cause an error.

After having worked with PLGS 2.3 for a bit over a years time, I am still running into weird error messages that I can not make any sense of. In the following subsections I will discuss various annoying details and bugs in PLGS. I have only included those bugs that I am able to reproduce and for which I have an idea about why they occur.

5.2.5.1 Searching Mascot from PLGS

PLGS has some problems with regard to submitting searches to Mascot. Since Mascot iteratively looks at lower and lower intensity ions, it is quite important not to include high intensity peaks that Mascot will not be able to match. PLGS has a couple of problems in this respect:

1. If the instrument was set to record the low m/z region (<160) then the spectra will include immonium ions. Mascot is able to handle immonium ions if configured to do so. This will be configured as a new instrument type. Unfortunately, in PLGS it is not possible to choose non-standard instruments from Mascot when setting up the Mascot search parameters. PLGS only shows a subset of instruments. None of those instruments have the optimal settings for the peaks included in the spectra PLGS sends to Mascot. So if the data includes immonium ions, then one of the standard Mascot instruments that can be chosen in PLGS must be reconfigured.

2. Because of the impossibility to choose an optimal activation energy with MS^E, PLGS will include parent ions in the peaks lists it submits to Mascot. But Mascot does not look for parent ions in the peak lists, so they will have a negative impacts on the scoring. As far as I have observed, the parent ions in MS^E data are often amongst the most intense ions in the spectra.

When viewing a peptide match in Mascot it displays which ions have been matched, from those ion series activated for the instrument. However, this ion match information was not saved from when the actual identification was made, but it is generated anew when opening the peptide match view. This also means that the information is not exported. Since the ion match information is not exported, PLGS generates that information itself when showing identifications from Mascot. Because another algorithm is used for finding the ions matches, the output will be very different. A big difference will typically be seen in the amount of low intensity ions that are matched. Apparently the algorithm in PLGS is not quite as picky as Mascot about which ions to include in the match. So if you are interested in which ions the match was based on, you have to open the Mascot search page and find it.

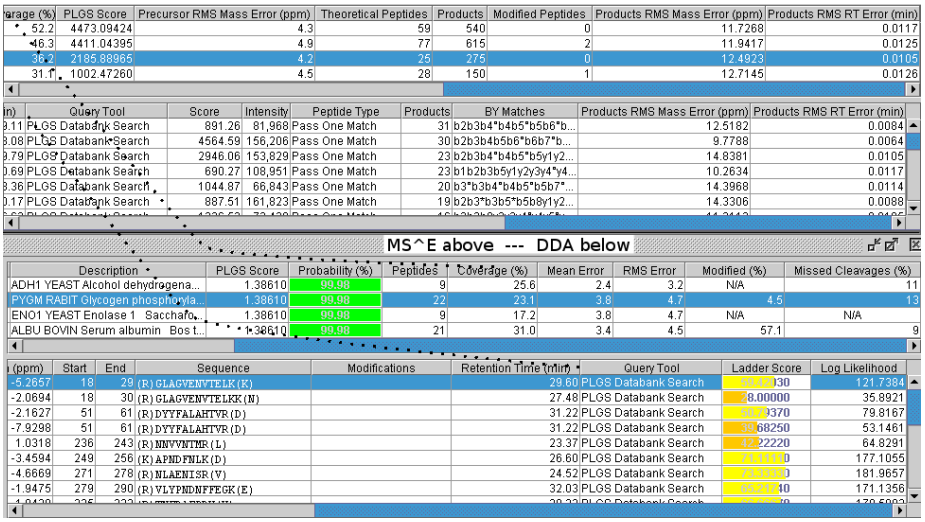


Figure 5.3: PLGS identification views for DDA data and MS^E data. The two tables at the top are protein matches and the peptide matches of the chosen protein for MS^E, while the bottom two tables are for DDA. The columns different for MS^E and DDA are placed to the right of the columns connected with the dotted lines. The picture is a screen shot from PLGS.

The peptide summary report is similar to that in Mascot, when viewed in PLGS. PLGS imports all peptide matches regardless of Mascot score. However, it does not mark the low scores with a negative OK filter even though the PLGS manual states that OK assignments will be set according to the Mascot scores. It is possible to sort the peptides according to Mascot score and then manually mark the low scoring peptides, so that the OK filter can be used to exclude low Mascot score hits, but this requires much additional effort.

I have also discovered a bug in the way PLGS imports Mascot results. When the same peptide is identified more than once (because of multiple charge states for instance), then the best scored identification of those matches is not shown in PLGS.

5.2.5.2 Identification when using MS^E compared to DDA

Searches in PLGS (only version 2.3) using MS^E and DDA differ a lot (see figure 5.3). These differences makes it difficult to compare MS^E and DDA identifications. In general MS^E views contains more information than DDA views, but the primary scoring measures of the peptides and proteins are also very different.

When searching DDA data the output will contain a *PLGS Score* and a *Probability* for proteins, and a *Ladder Score* and a *Log Likelihood* for peptides. With MS^E data PLGS will only include a protein *Score* and a peptide *Score*. The manual describes how the DDA scores are generated and how MS^E data is searched, but not how MS^E scores are generated. However, by looking at a simple example of a search on the four protein mpds database, it is obvious that

the two protein scores are not comparable at all. In this search, the top scoring protein of the MS^E data gets a score of 4473. By definition, the DDA score will never become larger than ln(#proteins in database) = ln(4) ≈ 1.39.

With the change of scoring mechanisms, MS^E also lost a feature they call auto curation. With DDA data (and in PLGS 2.2.5 also for MS^E data) the peptide and protein identifications below a certain threshold were shown, but marked by the auto curation as either bad, maybe, or ok. For MS^E data in PLGS 2.3, there are no such guidelines for when identifications are on the edge of being ok.

In addition to the differences between the primary scoring measures, MS^E views contains several columns that would be nice to have added to DDA views. Such as theoretical peptides, products, and the products rms error (see figure 5.3).

5.2.5.3 Identification scores using MS^E

During my evaluation I have come across some peptides for which the MS^E identification score does not seem to correspond with the quality of the match, an example of this can be seen in figure 5.4. In this example the peptides have the scores 1214.51 and 0.00 and the peptide with the high score has fewer product ions matched and the products rms mass error is worse. The only quality measure that the lowest scoring peptide has worse is the products rms retention time error, which is 0.0042 as opposed to 0.0030.

5.2.5.4 Label-free quantitation without MS^E

Another possibility for label-free quantitation (besides MS^E) is to use MS^1 data. PLGS version 2.2.5 can use this data for quantitation, and a separate DDA run can be used to assign identifications. I have not tested this possibility in detail, other than it is not possible in version 2.3, because of an error that makes version 2.3 think the data has been processed in an older version.

5.2.5.5 Regional settings

I have observed a problem with PLGS 2.3 in Windows XP Pro, when the regional setting are danish. Apparently, the regional settings must be English (or another using comma as decimal delimiter). I have been able to run PLGS 2.2.5 in Windows Vista with regional settings set to danish, with no problems (concerning regional settings) observed.

In PLGS 2.3, having the regional settings set to danish will cause some strange effects, but no obvious error messages. When searching the spectra in PLGS it will seem that it never finds any results. However, if you open the log files, you will see error messages denoting that a number format is wrong. Opening the project file (located in plgs2.3\root\project_number\more_obscure_numbers), it is possible to see that a search result has indeed been found.

This programming error happens because Java uses the regional settings when converting decimal numbers to strings. Apparently, the programmer has remembered to specify a locale when reading the files, but not when writing

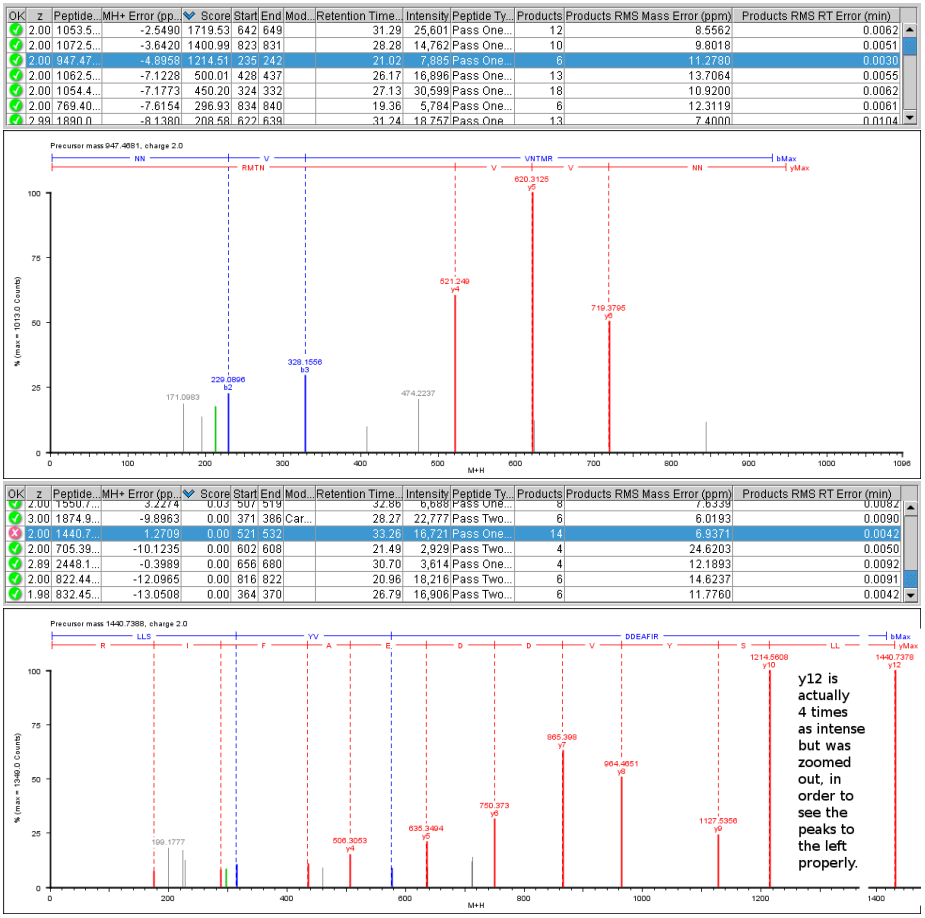


Figure 5.4: Two identified peptides from PLGS 2.3 using MS^E data. For each peptide, I have shown the key measures in the peptide table and the spectrum annotation. The top peptide is the one with the highest score and the bottom one has the lowest score. The pictures are screen shots from PLGS.

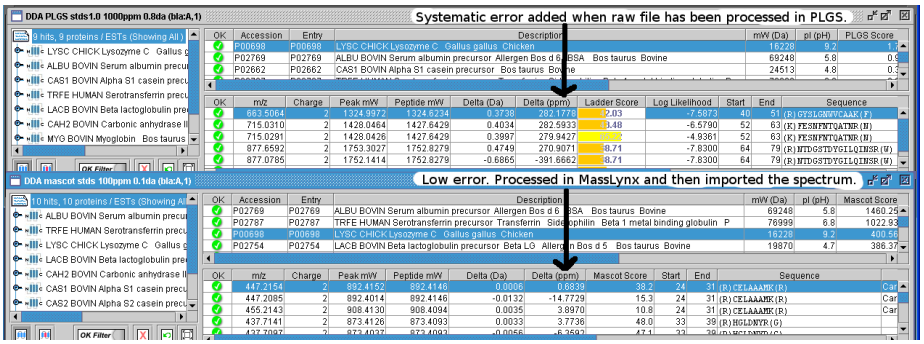


Figure 5.5: Mass shift in PLGS. The bottom one is searched through Mascot because PLGS does not handle pkl files containing precursors with higher than 1+ charge as described in section 5.2.5.7. The picture is a screen shot from PLGS.

them. Files that are written with the purpose of being read by a program and not a person should always be written (and read) with a specified locale (or the locale used should be written to the file also) in order to avoid these kinds of problems.

5.2.5.6 Shift error with DDA data from Synapt HDMS

Do not record DDA data with MassLynx configured to do real-time lockmass calibration. PLGS does not know how to handle this kind of data. As a result it shifts the mass values of DDA data. When processed with PLGS the data is shifted to one side, making the identifications ≈ 300 ppm off. If I process the same data with either MassLynx or msInspect the identifications are < 10 ppm off. This error occurs in both version 2.3 and 2.2.5 on the dataset I tested with.

According to correspondence with Waters, the data is being calibrated twice. This happens because the instrument was set to do real-time lockmass calibration at recording time, and then PLGS re-calibrates the data, because PLGS does not detect that the instrument has already calibrated the data. I have tested two workarounds for the problem: 1) I process the data with MassLynx. This creates pkl files that I can import into PLGS. 2) Waters suggested to manually set the calibration information in the dataset to 0, that way PLGS won't change anything when it tries to re-calibrate. While both solutions worked in terms of identifying the sample contents, it also made me discover two other bugs.

5.2.5.7 Searching imported peak lists with PLGS

Searching imported peak lists with PLGS is flawed, not in the search engine, but in the viewer that shows the results. Normally DDA data shows these five columns amongst others: The measured m/z, the assigned charge, the peak mW (which is $\frac{m}{z} * \text{charge}$), the theoretical peptide mW, and Delta (the difference between peak mW and peptide mW). However with imported peak lists, the

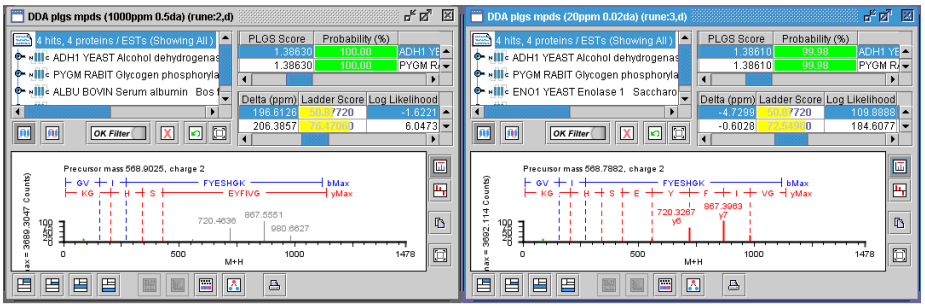


Figure 5.6: Annotation of two almost identical spectra with the same ladder score. The top spectrum is calibrated correctly, while the bottom spectrum is calibrated wrongfully. The annotations on the bottom spectrum lacks a lot of ions, but the ladder score is just as high as the top spectrum, indicating the same amount of ions were matched in the spectra. This happens because the shown annotations are generated independently of the search algorithm. The annotations use a hard limit of 0.1 Da, whereas my search used 0.5 Da. The pictures are screen shots from PLGS.

viewer forgets to multiply the m/z with the charge in the peak mW. The Delta column will then display $\frac{m}{z} * (\text{charge} - 1) + \text{real delta}$, when it should only show the real delta.

In order to be able to use this approach, I implemented a simple converter for turning all the precursors into their singly charged species. This is accomplished by:

$$\text{Singly charged } \frac{m}{z} = \frac{m}{z} * \text{charge} - (\text{hydrogen atom} - \text{electron}) * (\text{charge} - 1)$$

5.2.5.8 Spectrum annotations on DDA data

I have processed the same raw files twice. One time without fixing the calibration issue mentioned in section 5.2.5.6 and one where I changed the calibration information as in workaround two. Both give the same ladder scores, which should mean the same ions are matched, but the spectrum annotations are very different. As can be seen in figure 5.6, the unfixed version has only five ions annotated, while the fixed version has nine ions annotated.

The inconsistency between ladder scores and annotations occurs because the search engine does not save, which fragment ions it matched, with the rest of the search results. This information is recalculated by the viewer when it presents the search results. Apparently, the viewer does not take the mass tolerance used in the search in to account when it annotates the spectra. The viewer seems to have a tolerance of 100 mDa in version 2.3 and 15 mDa in version 2.2.5. The flaw in the spectrum annotations goes both ways, so if you have a narrow tolerance, then some annotations will be shown, even though they are outside your tolerance.

This flaw essentially makes the spectrum annotations useless for manual validation unless you have a tolerance of exactly 0.1 Da. However, if you just want

to use the annotated spectrum in an article or similar, and your tolerance is ≤ 0.1 Da, then you just need to check that none of the annotations are outside your tolerance[14].

5.2.5.9 SILAC labeling only somewhat supported

Since SILAC labeling modifies the mass of certain amino acids (see section 4.1.1) the search engine needs to know the new masses of the affected amino acids, because it will affect both the precursors mass and the masses of it's fragments. Furthermore, the quantitation algorithm needs to know the mass of the label to to select which precursors to compare.

In PLGS the way to do SILAC is to configure modifiers with the `quantitation reagent` parameter set to isotopic, assign those modifiers to samples (representing the conditions being compared), and then tell PLGS to make a combined sample for the conditions.

The modifiers have the following parameters: `Name`, `modifier type`, `quantitation reagent`, `delta mass`, `applies to`, and `fragments`. `Modifier type` defines whether it modifies the c-term, n-term, and/or sidechain. `Applies to` defines which AAs are affected. `Delta mass` defines how much heavier the AAs in `applies to` become. `Fragments` defines fragment ions resulting from this modifier.

Suppose I have a label that modifies both Arginine (R) and Lysine (K) by replacing all their carbon atoms with ^{13}C. This modifier is premade in PLGS, it is a sidechain `modifier type` with `delta mass` 6.020129 that `applies to` R and K.

Now suppose I have a label modifying K and R again, this time by replacing both carbon atom and nitrogen atom with heavier isotopes, namely ^{13}C and ^{15}N. This is not premade in PLGS, and it cannot be created in PLGS. The problem is that K and R does not have equal amounts of nitrogen atoms, so it would not be the same `delta mass` that apply to both K and R. Since each sample can only be assigned one modifier, it is impossible to quantitate using this label.

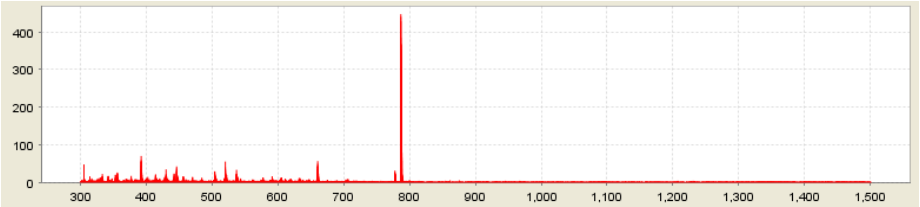
5.2.5.10 IP changes on laptops[15]

PLGS supports using a remote computer for processing the data (the data should then be located on that remote computer). However, is seems this feature creates a problem when using a laptop. Laptops often change their IPs depending on which network they are on, and in addition they will probably connect to VPN network which further confuses PLGS. It seems PLGS gets confused because it uses the IP address as part of the path to the raw data file.

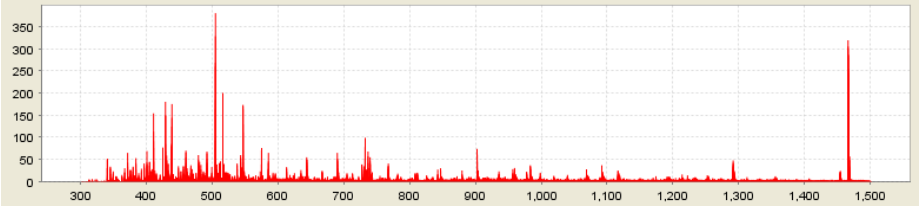
5.2.6 Waters MassLynx

MassLynx is the software with which you operate Waters instruments. The data visualization and processing features reflect this purpose, in that they are

[14] Easily done for a single spectrum, using their Fragment Ion Display.
[15] I only tested this issue in PLGS version 2.2.5



(a) Normal lockspray scan



(b) Strange lockspray scan

Figure 5.7: First (a) and second (b) scan in the lockspray function, all scans ought to look a lot like (a). Every other scan looks like (b) though. The pictures have been created using msInspect.

mostly oriented at processing a single spectrum at a time. Though it is possible to process an entire LCMS run into a peak list with one of the algorithms and it also possible to extend the features of MassLynx by installing add-ons.

I have not had many problem with MassLynx, but two issues are worth reporting. 1) Sometimes MassLynx does not show the data properly (it shows empty spectra), in particular, I have observed it with the "weird" lockspray spectra described in section 5.2.7. 2) The MassLynx installation routine must be installed in the default location. If installed in for instance `C:\MassLynx_test` errors will pop up at the end of the installation about some files not being able to register. Then if you click ok, the installer will tell you that the installation was successful despite the error just seen. However, the program will not work, and re-installation in the default folder is necessary to make it work.

5.2.7 Lockspray issues when running MS^E

On the Premier, we have observed that the lockspray function sometimes behaves very strange. The scans in the lockspray function can be divided in two groups, a normal group and a strange group. Every other scan will be normal and the other half will be strange, as can be seen in figure 5.7.

We have observed this only when recording MS^E, it does not occur when recording in DDA mode it seems. Waters supplied a workaround: It seems that the problem only arise when the two functions' mass ranges have unequal lower limits. So the function must have their lower mass ranges set to an equal value (and not set up as in the experiment run by Silva et al. [38]).

PLGS version 2.2.5 is able to process this data, by setting the amount of lockspray scans to combine above three. With PLGS version 2.3 it is not possible

to process the data. As was mentioned in section 5.2.5, Waters has simplified the set of parameters for MS^E data in PLGS version 2.3. Unfortunately, the simpler parameter settings in version 2.3 comes at a price. Waters has removed the possibility to combine lockspray scans in PLGS version 2.3, so in version 2.3 it is not possible to process MS^E data with a badly behaving lockspray function.

5.3 Sub-conclusion

I was my goal to identify a candidate open source software project that fulfilled the following criteria: It should be well maintained, easy to use, and fairly easy to add MS^E processing to. Moreover, in order to be able to test MS^E, the software should also have some support for label-free quantitation.

I identified five interesting open source projects to keep an eye on in the future: MZmine[2], xcms[14], OpenMS[10, 41], LabKey[1], and msInspect[8, 16]. MZmine will be releasing a new major version in the coming months. Xcms looks promising, but has problems with large data files. OpenMS has just released a new version, that I have been able to test yet, but a major improvement for many users will be that it is now available in Windows. All these three programs are designed for processing LCMS data and in particular, designed to do quantitation of LCMS data.

LabKey is a very professional data analysis system with support for flow cytometry, assays, and observational studies in addition to proteomics MS^2 analysis. It is open source software, with the option to buy support for an installation of the system. It is my interpretation, that LabKey will extend its interface with msInspect to bring label-free quantitation into LabKey in the near future. This will improve the already very useful application significantly. I recommend further investigating of how many in the Protein Research Group would benefit from a LabKey installation.

msInspect has been used extensively throughout this project. I have also contributed to the development of msInspect by fixing bugs, adding MS^E support, and adding a lockmass calibration module. msInspect has primarily been built for MS^1 data analysis and now extended to MS^E data too. The two main features of this program are its feature finding algorithm and its AMT module. Although I did not test the AMT module.

As is seen in chapters 3 and 4, my additions to msInspect are not enough to get proper results from MS^E data, the program to use is still PLGS. I have found and documented many problem and pitfalls with the PLGS software, however, if the issues with user friendliness is set aside, then PLGS performs very well at its core functions, which are: processing, searching, and quantitating MS^E data (and DDA data, although I would prefer using another search engine for DDA data).

Finally, I have also worked with the converter program MassWolf. MassWolf is the converter for Waters raw files →mzXML/mzML. I added support for converting MS^Eraw files to mzXML/mzML and proper support for converting the locksprayer function into mzXML/mzML.

Chapter 6

Project conclusion

The two major goals of this project were: 1) Investigate the performance of the fragmentation method MS^E in mass spectrometers developed from Waters Corporation, and 2) incorporate support for MS^E data in open source software projects.

MS^E support was added to the open source program msInspect. I used this in the investigation of MS^E by comparing the performance of msInspect to that of Waters data analysis program PLGS. The expectation was not that msInspect would perform better than PLGS, but that a proof of concept support for MS^E in msInspect could be the steppingstone for further improvements in the open source support for MS^E data. With some added effort in development of search engines, that can handle fragment spectra with multiple precursors specified, and additional improvements of the msInspect algorithms, this could become a viable platform for analyzing MS^E data. It is my hope that my efforts will aid development, when someone continues this work.

The evaluation of MS^E was overall positive, it performed slightly better than DDA, with respect to identification, and with respect to quantitation of protein ratios, it had very high accuracy. However, it should be noted the precision estimate of PLGS, the standard deviance, underestimates the error of its ratio estimate. For instance, the correct ratio of protein `ADH1_YEAST` is 1 and PLGS estimates a ratio of 1.08, with a standard deviance of 0.03, which makes the correct ratio ≈ 2.5 standard deviances away. This underestimation of the error also sneaks into the probability of up-regulation, that PLGS calculates. For the `ADH1_YEAST` protein, PLGS calculates a probability of up-regulation of 1.00, while in reality this protein is neither up or down regulated.

The comparison of DDA versus MS^E, was based on a sample consisting of four proteins. For this simple sample, MS^E outperformed DDA with 6.6% more peptides per protein, 19.7% more sequence coverage, and 9.9 more products per peptide corresponding to an increase of 162.1%. Only rms was slightly worse, it changed from 3.43 to 4.64 ($\uparrow$35.2%). Since the MS^E and DDA methods were compared with a very simple sample, it should be investigated how the results are affected, when the sample complexity is increased. I have investigated how MS^E performs with a complex sample, but I did not have any DDA recording of

that sample to compare with. Comparing with the MS^E of the complex sample to that of the simple sample, PLGS was able to identify about half as many of the peptides from the four proteins as it had with the simple sample, and with a lower quality of the peptide matches it did match.

While working with PLGS to generate the results for this report, I found and documented many problem and pitfalls with the PLGS software, however, if the issues with user friendliness is set aside, then PLGS performs very well at its core functions, which are: processing, searching, and quantitating MS^E data (and DDA data, although I would prefer using another search engine for DDA data).

As a final note, I would like to recommend further investigating of how many in the Protein Research Group would benefit from a LabKey installation. LabKey is a very professional data analysis system with support for flow cytometry, assays, and observational studies in addition to proteomics MS^2 analysis. In addition to being open source, it comes with the option to buy support for an installation of the system. It is my interpretation, that LabKey will extend its interface with msInspect to bring label-free quantitation into LabKey in the near future. This will improve the already very useful application significantly.

Chapter 7

Future perspectives

The natural next steps in continuing my work would be to: improve the comparisons with repeated experiments, improve the general and MS^E specific algorithms in msInspect, implement MS^E support in OpenMS, and to develop a search engine able to handle MS^E properly.

7.1 Comparisons

The comparisons need to be repeated a couple of times more in order to confirm the findings in this report. Furthermore, the comparisons of peptide identifications could be extended by using the quantitative information to locate false identifications and unidentified peptides, or by using the AMT feature in msInspect. Using the AMT feature might make it possible to increase the amount of identified features, and thus the amount of peptides to base the quantitation on, or, in the case of the complex +E.Coli sample, make quantitation possible in the first case.

It would be interesting to make a comparison of the results using different scan lengths for MS^E. The expected result would be that lowering the scan length would make it easier to separate the product features of parent features that have their elution maxima close to each other. The finer chromatographic granularity would come at the expense of lowered S/N, which then could be fought with an improvement in the feature finding algorithm, as described in section 7.2.2.

7.2 msInspect improvements

The single most important improvement to msInspect would be to increase the sampling frequency with which it resamples the spectra. After that I would improve the speed of the feature finding algorithm, and the visualization of the feature finding results. These topics will be further discussed in the following sections.

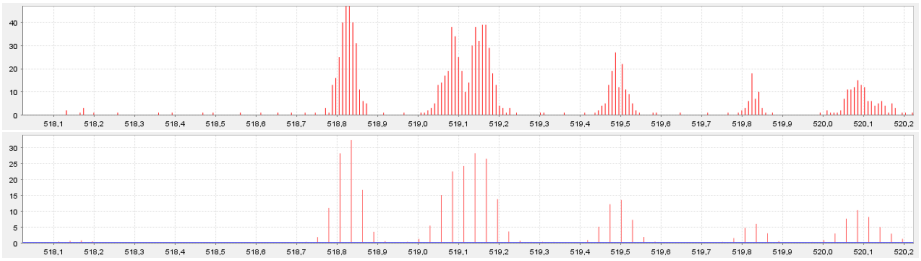


Figure 7.1: Raw data and resampled data using 36 bins per m/z unit. The pictures have been created using msInspect.

7.2.1 Resampling

msInspect resamples the data in 36 bins per m/z unit, this corresponds to a spacing of 0.0278 m/z between the bins (or 55.6ppm at 500 m/z and 18.5ppm at 1500 m/z). This is done both in order to make the algorithms run fast, and to make sure different scans have values at the same $\frac{m}{z}$. As can be seen in figure 7.1, this resampling lowers the resolution of the data, so that it is not possible to distinguish the two peaks (at 519.09 m/z and 519.15 m/z) from each other. The sampling frequency could improved by doubling it from 36 to 72 or another strategy could be used. For instance, more bins might be used in the lower m/z region and less in the upper region, such that the peak widths measured in bins would be constant over the entire $\frac{m}{z}$ range. However, this is rather complex to implement because many parts of msInspect rely on the frequency being 36. For instance, the wavelet function would need to be changed, to make it generate wavelets that match the new sampling frequency.

7.2.2 Feature finding

The feature finding algorithm processes every scan twice. This happens because it slides a feature finding window across the run. The width of this window is 256 scans, but the window is only moved 128 scans in each iteration. A lot of the processing that is executed twice, operate on a scan at a time and hence will generate identical results. So by reusing those parts of the results it should be possible to speed the computation up considerably.

Both with DDA and MS^E data fragment scans of the same parent ions are recorded more than once. With DDA this depends on the configuration of the instrument. This can be used to improve the signal to noise ratio by combining consecutive scans. I know that MassLynx normally does this combining (because they allow you to turn it off) and I expect PLGS to also do this (for DDA data), or else it would not really make any sense to record more than one scan of each parent ion (in DDA). My implementation of MS^E in msInspect does not take advantage of this, but care must be taken when implementing this to avoid slowing the processing too much.

Replicate	Parent ions	Product ions	Product ion scans
1	2	837	273
2	5	1007	234
3	13	1229	256

Table 7.1: Parent and product ions that were detected by the feature finder, but not matched to any products or parents respectfully. The MS^E dataset is described in section 3.2.2 as the Hye dataset. On average 254 scans containing at least one product ion does not have any neighboring parent ion to matched with. Those scans contain an average of 4 product ions. Some parent ions were also lost because of lacking product ions in the adjacent scans.

7.3 MS^E improvements

There are several aspects of the processing that might be improved. Both the feature finding, as described in section 7.2, the linking of parent and product ions, and the search engine.

Because of the need to use several activation energies with CAD, MS^E might work better with ETD, as ETD does not require different activation energies for different peptides. However, ETD require a charge state of minimum two and preferably three or more, and the fragmentation of ETD takes longer time than CAD to perform. Furthermore, ETD might not scale well to the increased amount of ions, in the sense that there might not be enough electrolytes to fragment all the ions. Nevertheless, if the time it takes for ETD to do its work does not destroy the chromatographic separation too much, it would be interesting to give it a shot.

7.3.1 Linking of parent and product ions

Additional work could be done for the MS^E featureset combiner described in section 5.2.4, I have some ideas for improving both the sensitivity and the specificity.

In order to improve the sensitivity, special attention should be put on the lonely features. Some parent ions won't have any detected product ions in their neighboring scans, and some product ions won't have any parent ions to be linked to. These lonely features should be investigated further, especially if several product ions peak in the same scan, but no parent ion have been detected in a neighboring scan. Some parent ions are also lost because of lacking product ions in the adjacent scans. However, it will often be easier to find the single parent ion for a set of product ion, than finding enough product ions for a parent ion, so the focus would be on those fragmentation scans in which a lot of product ions have been detected, but no adjacent parent ion.

For search engines such as Mascot, that does not cope well with fragments from multiple precursors, it is absolutely essential to increase the specificity of the linker. In order to get fragment spectra containing only fragments from a single parent ion, when the linker takes all the fragments from both the previous

and the next scan, would require the parent ions to have at least one survey scan with no parent ion in between them. There are two remedies for this: 1) Improve the granularity of the chromatography, either by changed the LC system or by decreasing the scan length. 2) Do not simply take all features from each neighboring fragment scan, but match the features on more attributes than only the scan number of the elution peak. For instance, the features could be matched using a model fitted to the shape of their elution profiles.

Using matching of a fitted elution model should be much easier to implement in OpenMS than msInspect, since OpenMS fits elution models to the features in some of it's feature finding algorithms.

7.3.2 Search engine

I have some ideas for improving the search efficiency. Waters describe their approach in the PLGS manual, a first try at improving the searching would naturally be to implement what they describe. Another way could be to re-submit the unmatched spectra from a Mascot run, but modify the unmatched spectra by removing the peaks that were found to belong to other peptides. In addition it would be interesting to test the open source program ProbIbTree that features identification of MS^2 spectra with multiple precursors. However, ProbIdTree has not been maintained since Zhang et al. [44] released the article describing it, so it might require some effort to get it to work.

7.3.3 User friendliness

MassWolf only has a command line interface, which many users will find hard to use. I would like to make it possible to use both msInspect and MassWolf through LabKey. It would greatly increase the user friendliness and it would make it possible to analyze label-free quantitation data from msInspect. These changes to LabKey would probably not take that much time, since it already contains code for interacting with both of these tools.

Bibliography

[1] Labkey / cpas. URL http://cpas.fhcrc.org.

[2] Mzmine. URL http://mzmine.sourceforge.net/.

[3] Pepper. URL http://www.broad.mit.edu/cancer/software/genepattern/desc/proteomics.html#pepper.

[4] Cpas documentation. URL https://www.labkey.org/wiki/home/Documentation/page.view?name=ms2.

[5] lcms2d. URL http://www.bioc.aecom.yu.edu/labs/angellab/.

[6] Mapquant. URL http://arep.med.harvard.edu/MapQuant/.

[7] Masswolf. URL http://tools.proteomecenter.org/wiki/index.php?title=Software:massWolf.

[8] msinspect. URL http://proteomics.fhcrc.org/CPL/msinspect.html.

[9] mzxml. URL http://sashimi.sourceforge.net.

[10] Openms. URL http://open-ms.sourceforge.net.

[11] Specarray. URL http://tools.proteomecenter.org/wiki/index.php?title=Software:SpecArray.

[12] Superhirn. URL http://tools.proteomecenter.org/wiki/index.php?title=Software:SuperHirn.

[13] Tpp. URL http://tools.proteomecenter.org/wiki/index.php?title=Software:TPP.

[14] Xcms. URL http://masspec.scripps.edu/xcms/xcms.php.

[15] J. M. Asara, H. R. Christofk, L. M. Freimark, and L. C. Cantley. A label-free quantification method by ms/ms tic compared to silac and spectral counting in a proteomics screen. *Proteomics*, 8(5):994–999, March 2008. ISSN 1615-9861. doi: 10.1002/pmic.200700426. URL http://dx.doi.org/10.1002/pmic.200700426.

[16] M. Bellew, M. Coram, M. Fitzgibbon, M. Igra, T. Randolph, P. Wang, D. May, J. Eng, R. Fang, C. Lin, J. Chen, D. Goodlett, J. Whiteaker,

A. Paulovich, and M. McIntosh. A suite of algorithms for the comprehensive analysis of complex protein mixtures using high-resolution lc-ms. *Bioinformatics*, 22(15):1902–1909, August 2006. ISSN 1460-2059. doi: 10.1093/bioinformatics/btl276. URL `http://dx.doi.org/10.1093/bioinformatics/btl276`.

[17] A. B. Chakraborty, S. J. Berger, and J. C. Gebler. Use of an integrated ms–multiplexed ms/ms data acquisition strategy for high-coverage peptide mapping studies. *Rapid Commun Mass Spectrom*, 21(5):730–744, 2007. ISSN 0951-4198. doi: 10.1002/rcm.2888. URL `http://dx.doi.org/10.1002/rcm.2888`.

[18] I. V. Chernushevich, A. V. Loboda, and B. A. Thomson. An introduction to quadrupole-time-of-flight mass spectrometry. *Journal of mass spectrometry : JMS*, 36(8):849–865, August 2001. ISSN 1076-5174. doi: 10.1002/jms.207. URL `http://dx.doi.org/10.1002/jms.207`.

[19] M. C. Codrea, C. R. Jiménez, J. Heringa, and E. Marchiori. Tools for computational processing of lc-ms datasets: a user's perspective. *Comput Methods Programs Biomed*, 86(3):281–290, June 2007. ISSN 0169-2607. doi: 10.1016/j.cmpb.2007.03.001. URL `http://dx.doi.org/10.1016/j.cmpb.2007.03.001`.

[20] Waters Corporation. Extending the quantitative linear range of a quadrupole-time of flight mass spectrometer using digital deadtime correction (ddtc). Waters Application Brief, October 2001. Version 1.

[21] Edmond de Hoffmann and Vincent Stroobant. *Mass Spectrometry, Principles and Applications*. Wiley, second edition, 1999.

[22] A. R. Dongre, J. L. Jones, A. Somogyi, and V. H. Wysocki. Influence of peptide composition, gas-phase basicity, and chemical modification on fragmentation efficiency: Evidence for the mobile proton model. *J. Am. Chem. Soc.*, 118(35):8365–8374, September 1996. doi: 10.1021/ja9542193. URL `http://dx.doi.org/10.1021/ja9542193`.

[23] Peicheng Du, Rajagopalan Sudha, Michael B. Prystowsky, and Ruth H. Angeletti. Data reduction of isotope-resolved lc-ms spectra. *Bioinformatics*, 23(11):1394–1400, June 2007. doi: 10.1093/bioinformatics/btm083. URL `http://dx.doi.org/10.1093/bioinformatics/btm083`.

[24] Clemens Gröpl, Eva Lange, Knut Reinert, Oliver Kohlbacher, Marc Sturm, Christian G. Huber, Bettina M. Mayr, and Christoph L. Klein. Algorithms for the automated absolute quantification of diagnostic markers in complex proteomics samples. pages 151–162. 2005. doi: 10.1007/11560500_14. URL `http://dx.doi.org/10.1007/11560500_14`.

[25] Jacob D. Jaffe, D. R. Mani, Kyriacos C. Leptos, George M. Church, Michael A. Gillette, and Steven A. Carr. Pepper, a platform for experimental proteomic pattern recognition. *Mol Cell Proteomics*, 5(10):1927–1941, October 2006. doi: 10.1074/mcp.M600222-MCP200. URL `http://dx.doi.org/10.1074/mcp.M600222-MCP200`.

[26] H. Lam, E. W. Deutsch, J. S. Eddes, J. K. Eng, N. King, S. E. Stein, and R. Aebersold. Development and validation of a spectral library searching method for peptide identification from ms/ms. *Proteomics*, 7(5):655–667, March 2007. ISSN 1615-9853. doi: 10.1002/pmic.200600625. URL `http://dx.doi.org/10.1002/pmic.200600625`.

[27] E. Lange, C. Gröpl, K. Reinert, O. Kohlbacher, and A. Hildebrandt. High-accuracy peak picking of proteomics data using wavelet techniques. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pages 243–254, 2006. ISSN 1793-5091. URL `http://view.ncbi.nlm.nih.gov/pubmed/17094243`.

[28] J. Listgarten and A. Emili. Statistical and computational methods for comparative proteomic profiling using liquid chromatography-tandem mass spectrometry. *Mol Cell Proteomics*, 4(4):419–434, April 2005. ISSN 1535-9476. doi: 10.1074/mcp.R500005-MCP200. URL `http://dx.doi.org/10.1074/mcp.R500005-MCP200`.

[29] D. May, M. Fitzgibbon, Y. Liu, T. Holzman, J. Eng, C. J. Kemp, J. Whiteaker, A. Paulovich, and M. McIntosh. A platform for accurate mass and time analyses of mass spectrometry data. *J Proteome Res*, 6(7):2685–2694, July 2007. ISSN 1535-3893. doi: 10.1021/pr070146y. URL `http://dx.doi.org/10.1021/pr070146y`.

[30] L. N. Mueller, M. Y. Brusniak, D. R. Mani, and R. Aebersold. An assessment of software solutions for the analysis of mass spectrometry based quantitative proteomics data. *J Proteome Res*, 7(1):51–61, January 2008. ISSN 1535-3893. doi: 10.1021/pr700758r. URL `http://dx.doi.org/10.1021/pr700758r`.

[31] Alexey I. Nesvizhskii and Ruedi Aebersold. Interpretation of shotgun proteomic data: The protein inference problem. *Mol Cell Proteomics*, 4(10):1419–1440, October 2005. doi: 10.1074/mcp.R500012. URL `http://dx.doi.org/10.1074/mcp.R500012`.

[32] Shao-En Ong, Blagoy Blagoev, Irina Kratchmarova, Dan B. Kristensen, Hanno Steen, Akhilesh Pandey, and Matthias Mann. Stable isotope labeling by amino acids in cell culture, silac, as a simple and accurate approach to expression proteomics. *Mol Cell Proteomics*, pages M200025–MCP200+, May 2002. doi: 10.1074/mcp.M200025-MCP200. URL `http://dx.doi.org/10.1074/mcp.M200025-MCP200`.

[33] D. N. Perkins, D. J. Pappin, D. M. Creasy, and J. S. Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20(18):3551–3567, December 1999. ISSN 0173-0835. doi: 10.1002/(SICI)1522-2683(19991201)20:18\<3551::AID-ELPS3551\>3.0.CO;2-2. URL `http://dx.doi.org/10.1002/(SICI)1522-2683(19991201)20:18\<3551::AID-ELPS3551\>3.0.CO;2-2`.

[34] S. Purvine, J. T. Eppel, E. C. Yi, and D. R. Goodlett. Shotgun collision-induced dissociation of peptides using a time of flight mass analyzer. *Proteomics*, 3(6):847–850, June 2003. ISSN 1615-9853. doi: 10.1002/pmic.200300362. URL `http://dx.doi.org/10.1002/pmic.200300362`.

[35] P. L. Ross, Y. N. Huang, J. N. Marchese, B. Williamson, K. Parker, S. Hattan, N. Khainovski, S. Pillai, S. Dey, S. Daniels, S. Purkayastha, P. Juhasz, S. Martin, M. Bartlet-Jones, F. He, A. Jacobson, and D. J. Pappin. Multiplexed protein quantitation in saccharomyces cerevisiae using amine-reactive isobaric tagging reagents. *Mol Cell Proteomics*, 3(12):1154–1169, December 2004. ISSN 1535-9476. doi: 10.1074/mcp.M400129-MCP200. URL `http://dx.doi.org/10.1074/mcp.M400129-MCP200`.

[36] Ole Schulz-Trieglaff, Rene Hussong, Clemens Gröpl, Andreas Hildebrandt, and Knut Reinert. A fast and accurate algorithm for the quantification of peptides from mass spectrometry data. pages 473–487. 2007. doi: 10.1007/978-3-540-71681-5_33. URL `http://dx.doi.org/10.1007/978-3-540-71681-5_33`.

[37] I. Shadforth, D. Crowther, and C. Bessant. Protein and peptide identification algorithms using ms for use in high-throughput, automated pipelines. *Proteomics*, 5(16):4082–4095, November 2005. ISSN 1615-9853. doi: 10.1002/pmic.200402091. URL `http://dx.doi.org/10.1002/pmic.200402091`.

[38] J. C. Silva, R. Denny, C. A. Dorschel, M. Gorenstein, I. J. Kass, G. Z. Li, T. Mckenna, M. J. Nold, K. Richardson, P. Young, and S. Geromanos. Quantitative proteomic analysis by accurate mass retention time pairs. *Anal. Chem.*, 77(7):2187–2200, April 2005. doi: 10.1021/ac048455k. URL `http://dx.doi.org/10.1021/ac048455k`.

[39] Jeffrey C. Silva, Richard Denny, Craig Dorschel, Marc V. Gorenstein, Guo-Zhong Li, Keith Richardson, Daniel Wall, and Scott J. Geromanos. Simultaneous qualitative and quantitative analysis of theescherichia coli proteome: A sweet tale. *Mol Cell Proteomics*, 5(4):589–607, April 2006. doi: 10.1074/mcp.M500321-MCP200. URL `http://dx.doi.org/10.1074/mcp.M500321-MCP200`.

[40] Jeffrey C. Silva, Marc V. Gorenstein, Guo-Zhong Li, Johannes P. Vissers, and Scott J. Geromanos. Absolute quantification of proteins by lcmse: A virtue of parallel ms acquisition. *Mol Cell Proteomics*, 5(1):144–156, January 2006. doi: 10.1074/mcp.M500230-MCP200. URL `http://dx.doi.org/10.1074/mcp.M500230-MCP200`.

[41] Marc Sturm, Andreas Bertsch, Clemens Groepl, Andreas Hildebrandt, Rene Hussong, Eva Lange, Nico Pfeifer, Ole S. Trieglaff, Alexandra Zerck, Knut Reinert, and Oliver Kohlbacher. Openms - an open-source software framework for mass spectrometry. *BMC Bioinformatics*, 9(1), 2008. doi: 10.1186/1471-2105-9-163. URL `http://dx.doi.org/10.1186/1471-2105-9-163`.

[42] *PLGS 2.3 Manual*. Waters Corporation.

[43] Jon D. Williams, Michael Flanagan, Linda Lopez, Steve Fischer, and Luke A. Miller. Using accurate mass electrospray ionization-time-of-flight mass spectrometry with in-source collision-induced dissociation to sequence peptide mixtures. *Journal of Chromatography A*, 1020(1):11–26, December

2003. doi: 10.1016/j.chroma.2003.07.019. URL `http://dx.doi.org/10.1016/j.chroma.2003.07.019`.

[44] Ning Zhang, Xiao-Jun Li, Mingliang Ye, Sheng Pan, Benno Schwikowski, and Ruedi Aebersold. Probidtree: An automated software program capable of identifying multiple peptides from a single collision-induced dissociation spectrum collected by a tandem mass spectrometer. *PROTEOMICS*, 5 (16):4096–4106, 2005. doi: 10.1002/pmic.200401260. URL `http://dx.doi.org/10.1002/pmic.200401260`.

Appendix A

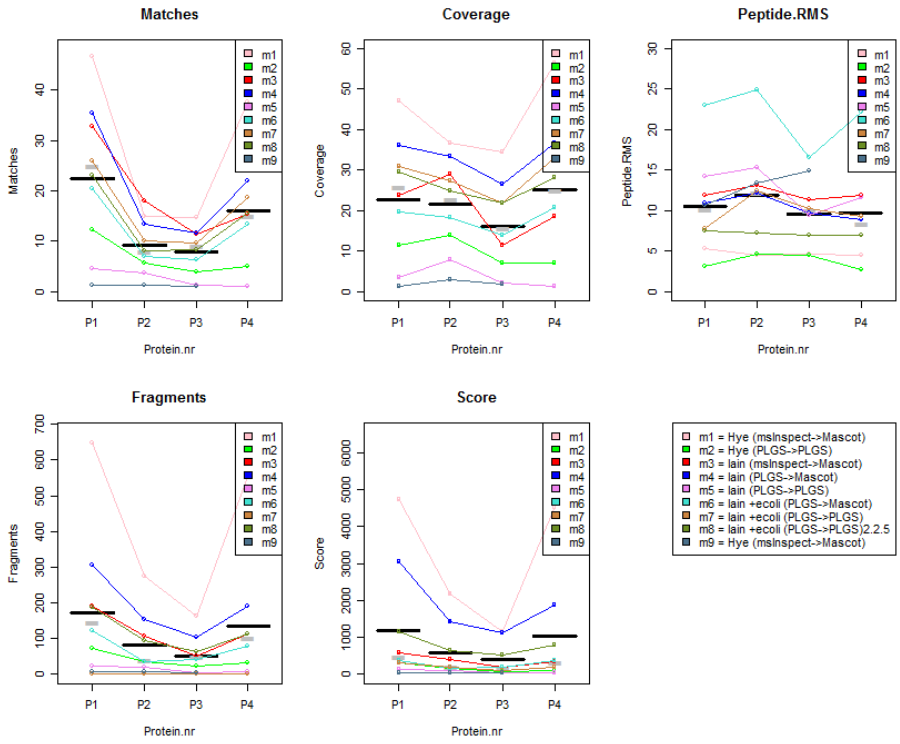
Statistical analysis

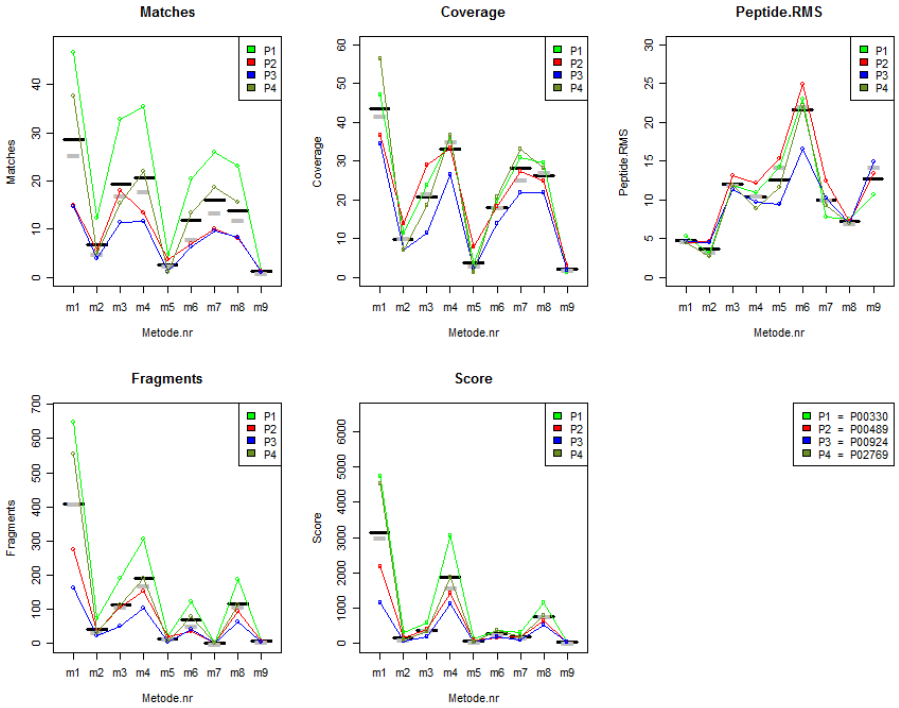
The following listing contains the anova output from R showing significant interaction between methods and proteins for all response variables but `Peptide.RMS`. For each response variable, the probability of the interaction between methods and proteins being random is listed in the column `Pr(>F)` at the row `method:Accession`.

```
Analysis of Variance Table
Response: Matches
                 Df Sum Sq Mean Sq  F value    Pr(>F)
method            8 6969.0   871.1  78.5374 < 2.2e-16 ***
Accession         3 3699.3  1233.1 111.1738 < 2.2e-16 ***
method:Accession 23 2043.7    88.9   8.0112 7.566e-12 ***
Residuals        69  765.3    11.1
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Analysis of Variance Table
Response: Coverage
                 Df  Sum Sq Mean Sq F value    Pr(>F)
method            8 16927.3  2115.9 89.4231 < 2.2e-16 ***
Accession         3   949.8   316.6 13.3804 5.607e-07 ***
method:Accession 23  1273.0    55.3  2.3391  0.003571 **
Residuals        69  1632.7    23.7
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Analysis of Variance Table
Response: Peptide.RMS
                 Df  Sum Sq Mean Sq F value    Pr(>F)
method            8 2690.74  336.34 55.0284 < 2.2e-16 ***
Accession         3   79.76   26.59  4.3498  0.007259 **
method:Accession 23  191.06    8.31  1.3591  0.164926
Residuals        69  421.74    6.11
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Analysis of Variance Table
Response: Fragments
                  Df  Sum Sq Mean Sq F value    Pr(>F)
method             8 1569793  196224 602.276 < 2.2e-16 ***
Accession          3  233301   77767 238.692 < 2.2e-16 ***
method:Accession  23  381652   16594  50.931 < 2.2e-16 ***
Residuals         69   22480     326
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Analysis of Variance Table
Response: Score
                  Df    Sum Sq   Mean Sq F value    Pr(>F)
method             8 105525487  13190686 85.1161 < 2.2e-16 ***
Accession          3  10545863   3515288 22.6832 2.501e-10 ***
method:Accession  23  25146243   1093315  7.0549 1.207e-10 ***
Residuals         69  10693128    154973
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The following two figures show plots of protein vs response and method vs response respectively. The black and grey bars show the mean and median response respectively.





As can be seen from the two preceding figures, most of the methods rank in the same order for all proteins. An example of an exception is methods m3 and m4 for proteins P2 and P4 with the response variable Matches. Using Matches, P2 ranks m3 better than m4, and P4 ranks m4 better than m3.

Appendix B

Decoy searching the mpds data on swissprot

Decoy searching is normally performed on a randomized or reversed database. However, since I have a sample of known content, I can use real protein databases as decoy. Any proteins identified, that are not one of the four mpds proteins are considered false positives. The results can be seen in figure B.1. Using the threshold of minimum 33 for peptide scores indicated by Mascot all the false positive results are filtered away.

Protein accession	Mascot score	Peptides matched
P00489	184	5
P00330	142	4
P02769	64	2
P00924	57	2
Q9QZY9	33	1
Q865F1	31	1
P35811	22	1
Q8CE50	20	1
Q3BAK5	19	1
P25847	18	1
Q8K352	18	1
Q8WVV4	18	1
P09116	17	1
Q4UKM1	17	1
O83142	16	1
P14164	16	1
Q5XLR4	16	1
P12385	16	1

Figure B.1: Mascot search results on one of the Iain MS^E files (processed with PLGS 2.3). Peptide scores >33 indicate identity or extensive homology ($p<0.05$).

Appendix C

Samples designed for testing MS^E

I designed a set of samples to test the MS^E performance, they were also prepared, unfortunately, they are still in the freezer because of instrument problems and time issues. However, I did acquire alternative samples to use for my analysis.

I designed my samples to test the influence of two parameters (complexity and dilution) on the performance of identification and one parameter (dilution) on the performance of quantitation. The following table shows how I designed the samples. Each of the samples have been made in three degrees of dilution, the concentration in the table, $^2/_3$, and $^1/_3$. Samples 1, 3.1, and 4 contain 1, 6, and 10 proteins respectively, and should be used for testing the influence of complexity on the performance of identification, while samples 2.{1,2,3} and 3.{1,2,3} are for quantitation comparisons.

The quantitation samples has been mixed separately three times for samples 2 and 3 in order to minimize effects of titration errors. I would run the samples in pairs of (2.1, 3.1), (2.2, 3.2), (2.3, 3.3), to be able to consider each pair as a protein ratio observation, and improving the statistics. If instrument time permits, each pair could be run more than once, further improving the statistics.

		Samples (concentration in fm/µl)							
MW	**Protein**	**1**	**2.1**	**2.2**	**2.3**	**3.1**	**3.2**	**3.3**	**4**
47260	Carbonic Anhydrase (bovin)		75	75	75	75	75	75	75
69248	BSA (bovin)	75	75	75	75	75	75	75	75
42854	Ovalbumin (chicken)		150	150	150	75	75	75	75
24479	Alpha casein (bovin milk)								75
25072	Beta casein (bovin milk)								75
19908	Beta lactoglobulin (bovin milk)		25	25	25	75	75	75	75
14398	RnaseB (bovin pancreas)								75
77010	Transferrin (human)		40	40	40	75	75	75	75
23232	Lysozyme (chicken)		10	10	10	75	75	75	75
17070	Myoglobin (whale)								75

Appendix D

Quantitation calculation methods

Protein	Median mean$^{\alpha}$	Mean std. dev.	Mean IQR	Mean median$^{\beta}$	Median std. dev.	Median IQR	Mean ignore peptides	Median ignore peptides
All peptides (max intensity)								
ADH1_YEAST	1.050	0.291	0.1310	1.110	0.276	0.1420	0.988	0.980
PYGM_RABIT	0.614	0.693	0.0801	0.762	0.715	0.0919	0.584	0.613
ENO1_YEAST	1.760	2.400	0.1940	2.670	2.550	0.1900	1.910	1.690
ALBU_BOVIN	7.090	14.900	3.2400	9.630	14.900	3.4700	5.090	6.420
Only those peptides that are identified in all runs								
ADH1_YEAST	1.060	0.295	0.1250	1.150	0.275	0.137	1.050	1.020
PYGM_RABIT	0.614	0.731	0.0831	0.779	0.755	0.111	0.597	0.631
ENO1_YEAST	1.760	2.500	0.1920	2.660	2.650	0.154	1.870	1.560
ALBU_BOVIN	7.090	3.140	2.9300	7.220	3.510	3.130	5.210	6.490
All peptides (intensity integrated over time)								
ADH1_YEAST	0.986	0.243	0.0859	1.070	0.268	0.1040	0.968	0.933
PYGM_RABIT	0.560	0.679	0.0755	0.721	0.632	0.0893	0.563	0.616
ENO1_YEAST	1.900	2.230	0.3590	2.500	2.120	0.3850	1.980	2.040
ALBU_BOVIN	6.930	17.200	2.3200	10.100	17.300	2.2200	5.640	6.770
Only those peptides that are identified in all runs								
ADH1_YEAST	1.00	0.263	0.1060	1.090	0.289	0.1050	0.996	0.979
PYGM_RABIT	0.56	0.716	0.0932	0.737	0.667	0.0884	0.571	0.622
ENO1_YEAST	1.90	2.310	0.3840	2.550	2.200	0.4110	1.930	2.060
ALBU_BOVIN	6.93	2.720	2.2700	7.140	3.110	2.1000	5.710	6.860

Two columns demand further explanation: α) Here I used the mean to calculate peptide ratios, and then took the median of those peptide ratios. β) The opposite of α, first median and the mean of the peptide ratios gave the protein ratio. The last two columns show the results from when the peptides are ignored and the protein ratios are simply the mean or median of all features identified for that protein.

Appendix E

Software - source code

Please note, this is not an exhaustive list of all source I touched during this project, and some of the code has been copied from the msInspect codebase. For instance, I have not included the changes I made to MassWolf, as I judged them unimportant.

The following source code is listed:

Listing E.1: Lockmass Calibration

```
1  package org.fhcrc.cpl.viewer.commandline.modules;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.text.DecimalFormat;
6  import java.text.NumberFormat;
7  import java.util.ArrayList;
8  import java.util.Arrays;
9  import java.util.List;
10
11 import org.fhcrc.cpl.viewer.MSRun;
12 import org.fhcrc.cpl.viewer.MSRun.MSScan;
13 import org.fhcrc.cpl.viewer.commandline.CommandLineModule;
14 import org.fhcrc.cpl.viewer.commandline.
       CommandLineModuleExecutionException;
15 import org.fhcrc.cpl.viewer.commandline.arguments.
       ArgumentValidationException;
16 import org.fhcrc.cpl.viewer.commandline.arguments.
       CommandLineArgumentDefinition;
17 import org.fhcrc.cpl.viewer.feature.Feature;
18 import org.fhcrc.cpl.viewer.feature.FeatureSet;
19 import org.fhcrc.cpl.viewer.feature.Smooth2D;
20 import org.fhcrc.cpl.viewer.feature.Spectrum;
21 import org.fhcrc.cpl.viewer.feature.extraction.DefaultPeakCombiner;
22 import org.fhcrc.cpl.viewer.feature.extraction.PeakCombiner;
23 import org.fhcrc.cpl.viewer.gui.util.PanelWithScatterPlot;
24 import org.fhcrc.cpl.viewer.gui.util.ScatterPlotDialog;
25 import org.jfree.chart.renderer.AbstractRenderer;
26 import org.jfree.chart.renderer.xy.StandardXYItemRenderer;
27 import org.jfree.data.xy.XYSeries;
28 import org.labkey.common.tools.ApplicationContext;
29 import org.labkey.common.tools.FloatRange;
30
31 public class CalibrationUsingLockspray extends BaseCommandLineModuleImpl
32   implements CommandLineModule {
33
34  File mzXmlFile, outFile;
35  FeatureSet featureSetFile;
36  float lockmassMz = 785.8426F;
37  int lockmassCharge = 2;
38  float massWindow = 0.2F;
39  boolean usePPM = false;
40  boolean plotCalibration = false;
41
42  public CalibrationUsingLockspray() {
43   init();
44  }
45
46  protected void init() {
47   mCommandName = "CalibrationUsingLockspray";
48
49   mHelpMessage = "Calibrate␣a␣FeatureSet␣or␣file␣using␣the␣scans␣with␣
         type=calibration␣in␣a␣mzXML␣raw␣file.";
50
51   mShortDescription = "Calibrate␣a␣FeatureSet␣file␣using␣type=
         calibration␣scans.";
52
53   CommandLineArgumentDefinition[] argDefs = {
54     createFileToReadArgumentDefinition("mzXmlFile", true,
55       "mzXML␣File"),
56     createFeatureFileArgumentDefinition("featureSetFile", false,
57       "FeatureSet␣File"),
58     createFileToWriteArgumentDefinition("featureout", false,
59       "Output␣file␣(featureSet)"),
60     createBooleanArgumentDefinition("plotCalibration", false,
61       "Plot␣the␣calibration", plotCalibration), };
62   addArgumentDefinitions(argDefs);
63  }
64
65  @Override
66  public void assignArgumentValues() throws ArgumentValidationException {
67   mzXmlFile = getFileArgumentValue("mzXmlFile");
68   featureSetFile = getFeatureSetArgumentValue("featureSetFile");
```

```
69    outFile = getFileArgumentValue("featureout");
70    plotCalibration = getBooleanArgumentValue("plotCalibration");
71   }
72
73   @Override
74   public void execute() throws CommandLineModuleExecutionException {
75    MSRun run;
76    try {
77     run = MSRun.load(mzXmlFile.toString());
78    } catch (IOException e) {
79     throw new CommandLineModuleExecutionException(
80       "Error processing file", e);
81    }
82    MSScan[] calibScans = run.getCalibrationScans();
83    if (calibScans == null || calibScans.length == 0) {
84     throw new CommandLineModuleExecutionException(
85       "No calibration scans in " + run.getFileName());
86    }
87
88    Feature[] lockmassFeatures = new Feature[calibScans.length];
89    for (int i = 0; i < calibScans.length; i++) {
90     float[][] spectrum = calibScans[i].getSpectrum();
91
92     // other processing
93     spectrum = Spectrum.ResampleSpectrum(spectrum, new FloatRange(
94       lockmassMz - 1, lockmassMz + 4), 36, false);
95     int window = 72;
96     float x[] = spectrum[1];
97     float bg[] = Spectrum.MinimaWindow(x, spectrum[0].length, window,
98       null);
99     for (int j = 0; j < bg.length; j++) {
100     bg[j] = Math.max(0, x[j] - bg[j]);
101    }
102    spectrum = new float[][] { spectrum[0], bg };
103    double s = Smooth2D.smoothYfactor;
104    spectrum[1] = Spectrum.FFTsmooth(spectrum[1], s, false);
105    Spectrum.Peak[] peaks = Spectrum.WaveletPeaksD3(spectrum);
106
107    PeakCombiner combiner = new DefaultPeakCombiner();
108    Feature[] features = combiner.createFeaturesFromPeaks(run, peaks);
109
110    float minDiff = massWindow;
111    Feature lockmassFeature = null;
112    List<Feature> thrownFeatures = new ArrayList<Feature>(
113      features.length);
114    for (Feature feature : features) {
115     if (feature.getCharge() != lockmassCharge) {
116      thrownFeatures.add(feature);
117      continue;
118     }
119     float diff = Math.abs(feature.getMz() - lockmassMz);
120     if (diff < minDiff) {
121      if (lockmassFeature != null) {
122       thrownFeatures.add(lockmassFeature);
123      }
124      lockmassFeature = feature;
125      minDiff = diff;
126     } else {
127      thrownFeatures.add(lockmassFeature);
128     }
129    }
130    lockmassFeatures[i] = lockmassFeature;
131   }
132
133   if (plotCalibration) {
134    ScatterPlotDialog spd = new ScatterPlotDialog();
135    XYSeries series = new XYSeries("Wavelet peaks, within "
136      + massWindow + " Da window");
137    double maxDiff = 0;
138    for (int index = 0; index < lockmassFeatures.length; index++) {
139     if (lockmassFeatures[index] != null) {
140      double time = calibScans[index].getDoubleRetentionTime();
141      double diff = lockmassFeatures[index].getMz() - lockmassMz;
142      if (Math.abs(diff) > maxDiff) {
```

```
143         maxDiff = Math.abs(diff);
144        }
145        series.add(time, diff);
146       } else {
147        series
148          .add(calibScans[index].getDoubleRetentionTime(),
149            null);
150       }
151      }
152      PanelWithScatterPlot panelWScatterPlot = spd
153        .getPanelWithScatterPlot();
154      panelWScatterPlot.addSeries(series);
155      StandardXYItemRenderer renderer = panelWScatterPlot.getRenderer();
156      renderer.setPlotLines(true);
157      NumberFormat decFormat = NumberFormat.getInstance();
158      panelWScatterPlot.setAxisLabels(
159        "Time",
160        "Mass Deviation (Da), " + decFormat.format(maxDiff)
161          + "Da=="
162          + decFormat.format(maxDiff / lockmassMz * 1000000)
163          + "ppm");
164      spd
165        .setTitle("Lockmass peaks within a " + massWindow
166          + " Da window");
167      spd.setVisible(true);
168     }
169
170     boolean cutEveryOther = false;
171     if (cutEveryOther) {
172      for (int i = 0; i < lockmassFeatures.length; i++) {
173       if (i % 2 == 1) {
174        lockmassFeatures[i] = null;
175       }
176      }
177     }
178
179     if (featureSetFile != null) {
180      Feature[] features = featureSetFile.getFeatures().clone();
181      Arrays.sort(features, new Feature.ScanAscComparator());
182      int i = 0;
183      while (i < lockmassFeatures.length && lockmassFeatures[i] == null) {
184       i++;
185      }
186      if (i == lockmassFeatures.length - 1) {
187       ApplicationContext.infoMessage("No lockmass features found");
188       return;
189      }
190      Feature before = lockmassFeatures[i];
191      // if feature is before the first found lockmass feature, then
192      // only use the first lockmass feature for correction.
193      Feature after = before;
194      for (Feature feature : features) {
195       int scanNum = feature.getScan();
196       // check if the feature has moved past the 'after' feature.
197       while (scanNum > after.getScan()) {
198        if (i < lockmassFeatures.length - 2) {
199         i++;
200         if (lockmassFeatures[i + 1] == null) {
201          continue;
202         }
203         before = after;
204         after = lockmassFeatures[i + 1];
205        } else {
206         // since feature is after the last found lockmass
207         // feature, then
208         // only use the last lockmass feature for correction.
209         before = after;
210         break;
211        }
212       }
213       float correction = lockmassMz
214         - (before.getMz() + after.getMz()) / 2;
215       if (usePPM) {
216        correction = correction / lockmassMz * feature.getMz();
```

```
217       }
218       feature.setMz(feature.getMz() + correction);
219       feature.updateMass();
220     }
221     try {
222       featureSetFile.save(outFile);
223     } catch (IOException e) {
224       ApplicationContext
225         .errorMessage("Error while trying to write file '"
226           + outFile + "'", e);
227     }
228   }
229  }
230 }
```

Listing E.2: Featurefinding class used for product features

```
1  package org.fhcrc.cpl.viewer.feature.extraction.strategy;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.Collection;
6
7  import org.fhcrc.cpl.viewer.MSRun;
8  import org.fhcrc.cpl.viewer.feature.ExtractMaxima2D;
9  import org.fhcrc.cpl.viewer.feature.Feature;
10 import org.fhcrc.cpl.viewer.feature.Spectrum;
11 import org.fhcrc.cpl.viewer.feature.extraction.DefaultPeakCombiner;
12 import org.fhcrc.cpl.viewer.feature.extraction.PeakExtractor;
13 import org.fhcrc.cpl.viewer.feature.extraction.SmootherCreator;
14 import org.fhcrc.cpl.viewer.feature.extraction.WaveletPeakExtractor;
15 import org.labkey.common.tools.FloatRange;
16 import org.labkey.common.tools.Scan;
17
18 public class FeatureStrategyMSe extends FeatureStrategyWindow {
19
20     protected boolean peakRidgeWalkSmoothed =
21         PeakExtractor.DEFAULT_PEAK_RIDGE_WALK_SMOOTHED;
22
23     protected int msLevel = 1;
24
25  protected WaveletPeakExtractor extractor = new WaveletPeakExtractor();
26  protected DefaultPeakCombiner peakCombiner = new DefaultPeakCombiner();
27
28
29     public FeatureStrategyMSe() {
30     }
31
32     public FeatureStrategyMSe(int msLevel) {
33      this();
34      this.msLevel = msLevel;
35     }
36
37     /**
38      * @param run
39      * @param startScanIndex
40      * @param scanCount
41      * @param maxCharge
42      * @param range
43      */
44     @Override
45     public void init(MSRun run, int startScanIndex,
46                      int scanCount, int maxCharge, FloatRange range,
47                          boolean plotStatistics)
48     {
49      super.init(run, startScanIndex, scanCount, maxCharge, range,
50          plotStatistics);
51      if (msLevel == 2) {
52       _scans = _run.getMS2Scans();
53       if (_scans.length > scanCount) {
54        int endScanIndex = Math.max(startScanIndex + scanCount, _scans.
55            length);
56        _scans = Arrays.copyOfRange(_scans, startScanIndex, endScanIndex)
```

```
                ;
54          }
55         }
56         extractor.setPeakRidgeWalkSmoothed(peakRidgeWalkSmoothed);
57         extractor.setShortPeak(2);
58         extractor.setUseIntensityCutoff(false);
59            peakCombiner.setMaxCharge(_maxCharge);
60            peakCombiner.setResamplingFrequency(_resamplingFrequency);
61       }
62
63       @Override
64   protected Collection<Feature> findPeptidesIn2DWindow(float[][] spectra,
65     Scan[] scanWindow) throws InterruptedException {
66         Feature[] allPeptides = extractor.extractPeakFeatures(_scans,
               spectra, _mzRange);
67   //   Spectrum.Peak[] rawPeaks =
68   //               ExtractMaxima2D.analyze(spectra, _mzRange.min, 1 / ((float
       ) _resamplingFrequency), SmootherCreator.getThresholdSmoother(),
       0.0F);
69   //
70     Arrays.sort(allPeptides, Spectrum.comparePeakMzAsc);
71           allPeptides = peakCombiner.createFeaturesFromPeaks(_run,
                 allPeptides);
72
73     java.util.List<Feature> features = new ArrayList<Feature>();
74   //   for (Spectrum.Peak rawPeak : rawPeaks) {
75   //     features.add(new Feature(rawPeak));
76   //   }
77     for (Feature peptide : allPeptides) {
78      features.add(peptide);
79     }
80     return features;
81   }
82
83   @Override
84   public boolean isPeakRidgeWalkSmoothed() {
85    return peakRidgeWalkSmoothed;
86   }
87
88   @Override
89   public void setPeakRidgeWalkSmoothed(boolean peakRidgeWalkSmoothed) {
90    this.peakRidgeWalkSmoothed = peakRidgeWalkSmoothed;
91   }
92
93 }
```

Listing E.3: Merely sets the MS level = 2, so that product features are extracted

```
1  package org.fhcrc.cpl.viewer.feature.extraction.strategy;
2
3  public class FeatureStrategyMSe_lvl2 extends FeatureStrategyMSe {
4   public FeatureStrategyMSe_lvl2() {
5    super(2);
6   }
7  }
```

Listing E.4: Combines Parent and Product features to create peak lists

```
1  package org.fhcrc.cpl.viewer.commandline.modules;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.IOException;
6  import java.io.PrintWriter;
7  import java.lang.reflect.Constructor;
8  import java.util.ArrayList;
9  import java.util.Arrays;
10 import java.util.HashMap;
11 import java.util.List;
12 import java.util.Map;
13 import java.util.Set;
14 import java.util.Map.Entry;
```

```
15
16 import org.fhcrc.cpl.viewer.MSRun;
17 import org.fhcrc.cpl.viewer.commandline.
       CommandLineModuleExecutionException;
18 import org.fhcrc.cpl.viewer.commandline.arguments.
       ArgumentValidationException;
19 import org.fhcrc.cpl.viewer.commandline.arguments.
       CommandLineArgumentDefinition;
20 import org.fhcrc.cpl.viewer.feature.Feature;
21 import org.fhcrc.cpl.viewer.feature.FeatureSet;
22 import org.fhcrc.cpl.viewer.gui.util.ChartDialog;
23 import org.fhcrc.cpl.viewer.gui.util.PanelWithBarChart;
24 import org.fhcrc.cpl.viewer.gui.util.PanelWithBlindImageChart;
25 import org.fhcrc.cpl.viewer.gui.util.PanelWithChart;
26 import org.fhcrc.cpl.viewer.gui.util.PanelWithHistogram;
27 import org.fhcrc.cpl.viewer.gui.util.PanelWithLineChart;
28 import org.jfree.chart.ChartFactory;
29 import org.jfree.data.statistics.HistogramBin;
30 import org.jfree.data.statistics.SimpleHistogramBin;
31 import org.jfree.data.statistics.SimpleHistogramDataset;
32 import org.jfree.data.xy.IntervalXYDataset;
33 import org.labkey.common.tools.ApplicationContext;
34
35 public class MSeFeatureSetCombinerCLM extends BaseCommandLineModuleImpl
36 {
37  File mzXmlFile, outFile, savePrecursors, saveFragments;
38  Map<Feature, Feature[]> pre_to_frag = new HashMap<Feature, Feature[]>()
       ;
39  FeatureSet precursors, fragments;
40  int inBothCounter, inNoneCounter, multipleParentsCounter;
41  int skippedFragmentsCounter, skippedFragmentScans, noFragmentsCounter;
42  int precursorsCount;
43  Map<Integer, Integer> fragsInSkippedScansHistogramData = new HashMap<
       Integer, Integer>();
44
45  /**
46   * Use the fragments from both neighboring scans of a precursor,
47   * instead of only the fragment scans that contain the precursor.
48   */
49  boolean useFragsFromBoth = true;
50  protected boolean plotStatistics = false;
51  /**
52   * The maker class that knows how to create a format.
53   * "MGF" or "PKL" for instance.
54   * See FormatMaker.makers for a list.
55   */
56  String maker = "MGF";
57
58  public MSeFeatureSetCombinerCLM()
59  {
60   init();
61  }
62
63  protected void init()
64  {
65   mCommandName = "msefeaturesetcombiner";
66
67   mHelpMessage = "Correlate␣precursor␣and␣fragment␣featuresets␣from␣a␣MS
       ^E␣experiment␣in␣order␣to␣get␣a␣'normal'␣DDA␣like␣peak␣list.";
68
69   mShortDescription = "Combine␣precursor␣and␣fragment␣featuresets␣from␣a
       ␣MS^E␣experiment.";
70
71   CommandLineArgumentDefinition[] argDefs =
72    {
73     createFileToReadArgumentDefinition("mzXMLfile", true,
74       "The␣mzXML␣file␣that␣the␣featuresets␣has␣been␣created␣from"),
75     createFeatureFileArgumentDefinition("precursors", true,
76       "featureset␣of␣the␣precursors"),
77     createFeatureFileArgumentDefinition("fragments", true,
78       "featureset␣of␣the␣fragments"),
79     createFileToWriteArgumentDefinition("outfile", true, "Output␣file"),
80     createBooleanArgumentDefinition("plotstats", false,
81                         "Plot␣statistics",
```

```
82                          plotStatistics),
83       createEnumeratedArgumentDefinition("formatMaker", false,
84         "Which format should be output?",
85         FormatMaker.makers, maker),
86      };
87    addArgumentDefinitions(argDefs);
88   }
89
90   @Override
91   public String toString()
92   {
93    return mCommandName + " " + mzXmlFile.getName();
94   }
95
96   @Override
97   public void assignArgumentValues() throws ArgumentValidationException
98   {
99    mzXmlFile = getFileArgumentValue("mzXMLfile");
100   precursors = getFeatureSetArgumentValue("precursors");
101   fragments = getFeatureSetArgumentValue("fragments");
102   outFile = getFileArgumentValue("outfile");
103   if (!hasArgumentValue("formatMaker"))
104   {
105    if (outFile.getName().toLowerCase().endsWith("pkl"))
106    {
107     maker = PKLMaker.makerName;
108    }
109    else if(outFile.getName().toLowerCase().endsWith("mgf"))
110    {
111     maker = MGFMaker.makerName;
112    }
113   }
114   else
115   {
116    maker = getStringArgumentValue("formatMaker");
117   }
118   plotStatistics = getBooleanArgumentValue("plotstats");
119         pre_to_frag.clear();
120         inBothCounter = 0;
121         inNoneCounter = 0;
122         multipleParentsCounter = 0;
123         noFragmentsCounter = 0;
124         skippedFragmentsCounter = 0;
125         skippedFragmentScans = 0;
126         precursorsCount = 0;
127      fragsInSkippedScansHistogramData.clear();
128  }
129
130  @Override
131  public void execute() throws CommandLineModuleExecutionException
132  {
133   PrintWriter outPW;
134   MSRun run;
135
136   try
137   {
138    outPW = getPrintWriter(outFile);
139   }
140   catch (FileNotFoundException e)
141   {
142    throw new CommandLineModuleExecutionException(e);
143   }
144
145   try
146   {
147    run = MSRun.load(mzXmlFile.getAbsolutePath());
148    if (run == null)
149    {
150     throw new CommandLineModuleExecutionException(
151       "Error opening run from file "
152         + mzXmlFile.getAbsolutePath());
153    }
154   }
155   catch (IOException e)
```

```
156     {
157      outPW.close();
158      throw new CommandLineModuleExecutionException(e);
159     }
160
161     Feature[] fragArray = fragments.getFeatures();
162     Arrays.sort(fragArray, new Feature.ScanAscComparator());
163     Feature[] precArray = precursors.getFeatures();
164     Arrays.sort(precArray, new Feature.ScanAscComparator());
165     int i = 0;
166     int prevScan = 0, scan = 0;
167     // Beware, I'm reusing this List Object, it changes with each
168     // precursor.
169     List<Feature> prevScanFrags = new ArrayList<Feature>();
170     List<Feature> nextScanFrags = new ArrayList<Feature>();
171     boolean inPrev = false, inNext = false;
172     boolean prevAssigned = false, nextAssigned = false;
173     int prevFragScanNum = 0, nextFragScanNum = 0;
174     /**
175      * use for finding fragmentation scans. run.getIndexForScanNum(scanNum
             ,
176      * true) where scanNum is a precursor scan number will give the scan
177      * index of the previous fragmentation scan.
178      */
179     run.setMsLevel(2);
180     for (Feature precursor : precArray) {
181      prevScan = scan;
182      scan = precursor.getScan();
183      inPrev = inNext = false;
184      if (prevScan != scan) {
185       int prevFragScanIndex = run.getIndexForScanNum(scan, true);
186       prevFragScanNum = run.getScan(prevFragScanIndex).getNum();
187       if (prevFragScanNum == nextFragScanNum) {
188        // Precursor only moved one scan number forward.
189        prevAssigned = nextAssigned;
190        nextAssigned = false;
191
192        // Put the fragments from the next list into the previous
193        // list.
194        List<Feature> tmp = prevScanFrags;
195        prevScanFrags = nextScanFrags;
196        nextScanFrags = tmp;
197        nextScanFrags.clear();
198        if (prevScanFrags.size() > 0) {
199         for (Feature frag : prevScanFrags) {
200          if (frag.getMass() == precursor.getMass()) {
201           inPrev = true;
202          }
203         }
204        }
205       } else {
206        prevAssigned = nextAssigned = false;
207        prevScanFrags.clear();
208        nextScanFrags.clear();
209        int lastSkippedScan = -2; // no scan numbers should match this.
210        int beforeSkippedFrags = skippedFragmentsCounter;
211        while (i < fragArray.length
212          && fragArray[i].getScan() < prevFragScanNum) {
213         // We are skipping some fragment features, throwing them
214         // away, because no precursor was found for them.
215         // XXX: Might want to record this, so that it's possible
216         // to target a search for precursors for the fragments.
217         if (fragArray[i].getScan() != lastSkippedScan) {
218          // First skipped fragScan should get in here.
219          skippedFragmentScans++;
220
221          int fragsInLastSkippedScan = skippedFragmentsCounter -
                 beforeSkippedFrags;
222          if (fragsInLastSkippedScan != 0) {
223           Integer count = fragsInSkippedScansHistogramData.get(
                  fragsInLastSkippedScan);
224           if (count == null) {
225            count = 0;
226           }
```

```
227             count += 1;
228             fragsInSkippedScansHistogramData.put(fragsInLastSkippedScan,
                    count);
229           } else {
230             assert lastSkippedScan == -2;
231           }
232           beforeSkippedFrags = skippedFragmentsCounter;
233           // Only get in here again when the skipped fragments are from
                another scan.
234           lastSkippedScan = fragArray[i].getScan();
235         }
236         skippedFragmentsCounter++;
237
238         i++;
239       }
240       int fragsInLastSkippedScan = skippedFragmentsCounter -
              beforeSkippedFrags;
241       if (fragsInLastSkippedScan != 0) {
242         Integer count = fragsInSkippedScansHistogramData.get(
              fragsInSkippedScansHistogramData);
243         if (count == null) {
244           count = 0;
245         }
246         count += 1;
247         fragsInSkippedScansHistogramData.put(fragsInLastSkippedScan, count
              );
248       }
249
250       while (i < fragArray.length
251         && fragArray[i].getScan() == prevFragScanNum) {
252         if (fragArray[i].getMass() == precursor.getMass()) {
253           inPrev = true;
254         }
255         prevScanFrags.add(fragArray[i]);
256         i++;
257       }
258     }
259     nextFragScanNum = run.getScan(prevFragScanIndex + 1).getNum();
260
261     while (i < fragArray.length
262       && fragArray[i].getScan() == nextFragScanNum) {
263       if (fragArray[i].getMass() == precursor.getMass()) {
264         inNext = true;
265       }
266       nextScanFrags.add(fragArray[i]);
267       i++;
268     }
269   } else {
270     for (Feature frag : prevScanFrags) {
271       if (frag.getMass() == precursor.getMass()) {
272         inPrev = true;
273       }
274     }
275     for (Feature frag : nextScanFrags) {
276       if (frag.getMass() == precursor.getMass()) {
277         inNext = true;
278       }
279     }
280   }
281   if (prevScanFrags.size() == 0 && nextScanFrags.size() == 0) {
282     noFragmentsCounter++;
283   }
284   if (useFragsFromBoth || inNext == inPrev) {
285     // Weird. inNext && inPrev could be true, but it's not good.
286     // Means the precursor was found among the fragments in both
287     // the previous and the next scan.
288     // Would probably mean that the fragments contain the
289     // precursor in multiple charge states and
290     // the charge states peaked in different scans.
291     if (inNext && inPrev) {
292       inBothCounter++;
293     } else if (!inNext && !inPrev) {
294       inNoneCounter++;
295     }
```

```
296
297       if (prevAssigned || nextAssigned) {
298        multipleParentsCounter++;
299       }
300       nextAssigned = prevAssigned = true;
301       Feature[] tmp = new Feature[prevScanFrags.size()
302         + nextScanFrags.size()];
303       int j = 0;
304       for (Feature feature : prevScanFrags) {
305        tmp[j++] = feature;
306       }
307       for (Feature feature : nextScanFrags) {
308        tmp[j++] = feature;
309       }
310       pre_to_frag.put(precursor, tmp);
311      } else if (inPrev) {
312       if (prevAssigned) {
313        multipleParentsCounter++;
314       }
315       prevAssigned = true;
316       pre_to_frag.put(precursor, prevScanFrags
317         .toArray(new Feature[0]));
318      } else {
319       if (nextAssigned) {
320        multipleParentsCounter++;
321       }
322       nextAssigned = true;
323       pre_to_frag.put(precursor, nextScanFrags
324         .toArray(new Feature[0]));
325      }
326     }
327     fragArray = precArray = null;
328     prevScanFrags = nextScanFrags = null;
329
330     Set<Map.Entry<Feature, Feature[]>> entries = pre_to_frag.entrySet();
331     precursorsCount = entries.size();
332
333     FormatMaker formatMaker;
334     try
335     {
336      Class<? extends FormatMaker> makerClass = (Class<? extends
           FormatMaker>) Class.forName(getClass().getName() + "$" + maker +
            FormatMaker.classSuffix);
337      Constructor<? extends FormatMaker> makerConstructor = makerClass.
           getDeclaredConstructor(new Class[] {getClass()});
338      formatMaker = makerConstructor.newInstance(this);
339     }
340     catch (Exception e)
341     {
342      throw new CommandLineModuleExecutionException("Unable␣to␣use␣the␣
           formatMaker␣" + maker, e);
343     }
344
345     formatMaker.init(outPW);
346     for (Map.Entry<Feature, Feature[]> entry : entries) {
347      Feature pre = entry.getKey();
348      Feature[] frags = entry.getValue();
349      formatMaker.query(pre, frags);
350     }
351     formatMaker.terminate();
352     outPW.close();
353
354     if (plotStatistics) {
355      String[] xval = new String[fragsInSkippedScansHistogramData.size()];
356      int[] yval = new int[fragsInSkippedScansHistogramData.size()];
357      int x = 0;
358      for (Entry<Integer, Integer> entry : fragsInSkippedScansHistogramData
           .entrySet()) {
359       xval[x] = entry.getKey().toString();
360       yval[x] = entry.getValue();
361       x++;
362      }
363      PanelWithChart pwh = new PanelWithBarChart(xval, yval, "frags␣in␣
           skipped␣scans");
```

```
364     ChartDialog chartdlg = new ChartDialog(pwh);
365     chartdlg.setVisible(true);
366    }
367   }
368
369   private static boolean massEquals(float mass1, float mass2) {
370    return Math.abs(mass1 - mass2) / mass1 * 1E6 < 1;
371   }
372
373   interface FormatMaker {
374    void init(PrintWriter pw);
375    void query(Feature pre, Feature[] frags);
376    void terminate();
377    public String getMakerName();
378    public static final String[] makers = {MGFMaker.makerName, PKLMaker.
         makerName};
379    public static final String classSuffix = "Maker";
380   }
381
382   class PKLMaker implements FormatMaker {
383    PrintWriter outPW;
384    public static final String makerName = "PKL";
385
386    public String getMakerName()
387    {
388     return makerName;
389    }
390
391    public void init(PrintWriter pw) {
392     outPW = pw;
393    }
394
395    public void query(Feature pre, Feature[] frags) {
396     if (frags.length == 0)
397     {
398      return;
399     }
400     outPW.println(pre.getMz() + "␣" + pre.getTotalIntensity() + "␣" + pre
           .getCharge());
401     for (Feature frag : frags) {
402      //search engines do not expect the precursor in the fragment scan
403      if (massEquals(frag.getMass(), pre.getMass())) {
404       continue;
405      }
406      outPW.println(frag.getMz() + "␣" + frag.getTotalIntensity());
407     }
408     outPW.println();
409    }
410
411    public void terminate() { }
412   }
413
414   class MGFMaker implements FormatMaker {
415    PrintWriter outPW;
416    public static final String makerName = "MGF";
417
418    public String getMakerName() {
419     return makerName;
420    }
421
422    public void init(PrintWriter pw) {
423     outPW = pw;
424     outPW.println("#inBothCounter␣=␣" + inBothCounter);
425     outPW.println("#inNoneCounter␣=␣" + inNoneCounter);
426     outPW.println("#precursorsCount␣=␣" + precursorsCount);
427     outPW.println("#multipleParentsCounter␣=␣" + multipleParentsCounter);
428     outPW.println("#skippedFragmentsCounter␣=␣" + skippedFragmentsCounter
           );
429     outPW.println("#skippedFragmentScans␣=␣" + skippedFragmentScans);
430     outPW.println("#noFragmentsCounter␣=␣" + noFragmentsCounter);
431    }
432
433    public void query(Feature pre, Feature[] frags) {
434     outPW.println("BEGIN␣IONS");
```

```
435     outPW.println("TITLE=" + pre.toString());
436     outPW.println("PEPMASS=" + pre.getMz() + "␣" + pre.getTotalIntensity
            ());
437     if (pre.getCharge() != 0) {
438      outPW.println("CHARGE=" + pre.getCharge() + "+");
439     }
440     outPW.println("SCANS=" + pre.getScanFirst() + "-" + pre.getScanLast()
            );
441     outPW.println("RTINSECONDS=" + pre.getTime());
442     for (Feature frag : frags) {
443      //search engines do not expect the precursor in the fragment scan
444      if (frag.getMass() == pre.getMass()) {
445       continue;
446      }
447      outPW.println(frag.getMz() + "␣" + frag.getTotalIntensity());
448     }
449     outPW.println("END␣IONS");
450     outPW.println();
451    }
452
453    public void terminate() { }
454   }
455  }
```

Listing E.5: Assigns Mascot identifications to features

```
1   /*
2    * Copyright (c) 2003-2007 Fred Hutchinson Cancer Research Center
3    *
4    * Licensed under the Apache License, Version 2.0 (the "License");
5    * you may not use this file except in compliance with the License.
6    * You may obtain a copy of the License at
7    *
8    *      http://www.apache.org/licenses/LICENSE-2.0
9    *
10   * Unless required by applicable law or agreed to in writing, software
11   * distributed under the License is distributed on an "AS IS" BASIS,
12   * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
         implied.
13   * See the License for the specific language governing permissions and
14   * limitations under the License.
15   */
16  package org.fhcrc.cpl.viewer.commandline.modules;
17
18  import org.fhcrc.cpl.viewer.commandline.*;
19  import org.fhcrc.cpl.viewer.commandline.arguments.
        ArgumentValidationException;
20  import org.fhcrc.cpl.viewer.commandline.arguments.
        CommandLineArgumentDefinition;
21  import org.fhcrc.cpl.viewer.commandline.arguments.
        DeltaMassArgumentDefinition;
22  import org.fhcrc.cpl.viewer.commandline.arguments.
        EnumeratedValuesArgumentDefinition;
23  import org.fhcrc.cpl.viewer.feature.FeatureSet;
24  import org.fhcrc.cpl.viewer.feature.Feature;
25  import org.fhcrc.cpl.viewer.feature.extraInfo.MS2ExtraInfoDef;
26  import org.fhcrc.cpl.viewer.feature.extraInfo.AmtExtraInfoDef;
27  import org.fhcrc.cpl.viewer.amt.*;
28  import org.fhcrc.cpl.viewer.MSRun;
29  import org.fhcrc.cpl.viewer.util.RegressionUtilities;
30  import org.fhcrc.cpl.viewer.gui.util.ScatterPlotDialog;
31  import org.fhcrc.cpl.viewer.align.Aligner;
32  import org.labkey.common.tools.*;
33  import org.labkey.common.util.Pair;
34  import org.apache.log4j.Logger;
35  import org.jfree.data.xy.XYSeriesCollection;
36
37  import java.util.*;
38  import java.io.File;
39  import java.io.IOException;
40
41
42  /**
```

```
43  */
44  public class MSeFeatureMatcher extends BaseCommandLineModuleImpl
45  {
46      protected static Logger _log = Logger.getLogger(
            FeatureSetMatcherCommandLineModule.class);
47
48      protected FeatureSet ms1Features = null;
49      protected FeatureSet ms2Features = null;
50      protected File outFile = null;
51      protected MSRun run1 = null;
52      protected double minPeptideProphet = 0;
53      protected boolean useTime = false;
54      protected boolean writeUnmatched = true;
55      protected boolean stripMultipleMS2 = true;
56
57  //    protected FeatureSet[] ms2FeatureSets;
58
59      protected File outUnmatchedMS2File = null;
60      protected File outAllMS2MarkedFile = null;
61
62      protected boolean showCharts = false;
63
64      protected Protein[] proteinsInFasta = null;
65
66      protected List<MS2Modification> ms2ModificationsForMatching;
67
68      public MSeFeatureMatcher()
69      {
70          init();
71      }
72
73      protected void init()
74      {
75          mCommandName = "msematchfeatures";
76          mShortDescription = "perform␣simple␣feature-matching␣for␣MS^E␣
                data";
77
78          mHelpMessage =
79                  "perform␣feature-matching␣between␣a␣MS^E␣feature␣file␣
                        and␣it's␣pepXML␣identification";
80
81          CommandLineArgumentDefinition[] argDefs =
82              {
83                      createFeatureFileArgumentDefinition("ms1features",
                            true,
84                              "MS1␣feature␣file"),
85                      createFeatureFileArgumentDefinition("ms2features",
                            false,
86                              "MS2␣feature␣file␣(usually␣pepXml)"),
87                      createDirectoryToReadArgumentDefinition("ms2dir",
                            false,
88                              "MS2␣feature␣file␣directory"),
89                      createFileToReadArgumentDefinition("mzxml",false,
90                              "mzXML␣file"),
91                      createFileToWriteArgumentDefinition("out", false,
92                              "Output␣File"),
93                      createDecimalArgumentDefinition("minpprophet", false
                            ,
94                              "Minimum␣PeptideProphet␣score",
                                    minPeptideProphet),
95                      createBooleanArgumentDefinition("writeunmatched",
                            false,
96                              "Write␣out␣unmatched␣features",
                                    writeUnmatched),
97                      createBooleanArgumentDefinition("stripmultiplems2",
                            false,
98                              "Strip␣subsequent␣MS2␣identifications␣for␣
                                    the␣same␣peptide␣out␣of␣the␣file␣when␣
                                    matching",
99                              stripMultipleMS2),
100                     createFileToWriteArgumentDefinition("outunmatchedms2
                            ", false,
101                             "Output␣File␣for␣unmatched␣MS2"),
102                     createFileToWriteArgumentDefinition("outallms2marked
```

```
                        ", false,
103                         "Output File for all MS2, with unmatched
                                having 'unmatched' in description"),
104                 createBooleanArgumentDefinition("showcharts", false,
105                         "show useful charts created when matching",
                                showCharts),
106                 createFastaFileArgumentDefinition("fasta", false,
107                         "Fasta database for matching"),
108                 createModificationListArgumentDefinition("
                        modifications", false,
109                         "a list of modifications to use when
                                creating features to represent peptide
                                sequences"),
110             };
111         addArgumentDefinitions(argDefs);
112     }
113
114     public void assignArgumentValues()
115             throws ArgumentValidationException
116     {
117         ms1Features = getFeatureSetArgumentValue("ms1features");
118
119         if (hasArgumentValue("minpprophet"))
120             minPeptideProphet = (float) getDoubleArgumentValue("
                    minpprophet");
121         FeatureSet.FeatureSelector peptideProphetFeatureSelector = new
                FeatureSet.FeatureSelector();
122         peptideProphetFeatureSelector.setMinPProphet((float)
                minPeptideProphet);
123
124         stripMultipleMS2 = getBooleanArgumentValue("stripmultiplems2");
125
126         if (hasArgumentValue("mzxml"))
127         {
128             try
129             {
130                 run1 = MSRun.load((getFileArgumentValue("mzxml")).
                        getAbsolutePath());
131             }
132             catch (Exception e)
133             {
134                 throw new ArgumentValidationException(e);
135             }
136         }
137
138         assertArgumentPresent("ms2features");
139         ms2Features = getFeatureSetArgumentValue("ms2features");
140         ms2Features = ms2Features.filter(peptideProphetFeatureSelector);
141         if (ms2Features.getFeatures().length == 0)
142             throw new ArgumentValidationException("There are no MS2
                    features with PeptideProphet score >= " +
                    minPeptideProphet + ", quitting.");
143         if (run1 != null)
144             ms2Features.populateTimesForMS2Features(run1);
145         if (stripMultipleMS2)
146             MS2ExtraInfoDef.removeAllButFirstFeatureForEachPeptide(
                    ms2Features);
147
148         outUnmatchedMS2File = getFileArgumentValue("outunmatchedms2");
149         outAllMS2MarkedFile = getFileArgumentValue("outallms2marked");
150
151         ApplicationContext.infoMessage("MS1 features: " + ms1Features.
                getFeatures().length);
152 //          ApplicationContext.infoMessage("MS2 features: " + ms2Features.
    getFeatures().length);
153
154         outFile = getFileArgumentValue("out");
155
156         writeUnmatched = getBooleanArgumentValue("writeunmatched");
157
158         showCharts = getBooleanArgumentValue("showCharts");
159     }
160
161
```

```
162     /**
163      * do the actual work
164      */
165     public void execute() throws CommandLineModuleExecutionException
166     {
167      matchOneToOne(ms1Features, ms2Features);
168     }
169
170     protected void matchOneToOne(FeatureSet ms1Features, FeatureSet
            ms2Features)
171             throws CommandLineModuleExecutionException
172     {
173         ms1Features.addExtraInformationType(MS2ExtraInfoDef.
                getSingletonInstance());
174
175         _log.debug("ms1Features: " + ms1Features.getFeatures().length +
                ", ms2Features: " + ms2Features.getFeatures().length);
176         Set<String> allMS2Peptides = new HashSet<String>();
177         for (Feature ms2Feature : ms2Features.getFeatures())
178         {
179             allMS2Peptides.add(MS2ExtraInfoDef.getFirstPeptide(
                    ms2Feature));
180         }
181
182
183         XYSeriesCollection dataset = new XYSeriesCollection();
184
185
186         AmtFeatureSetMatcher.FeatureMatchingResult featureMatchingResult
                =
187                 (new MyMatcher()).matchFeatures(ms1Features, ms2Features
                        );
188
189         Set<Feature> matchedMs1FeatureHashSet = new HashSet<Feature>();
190         Set<String> matchedPeptides = new HashSet<String>();
191         Set<Feature> matchedMs2FeatureHashSet = new HashSet<Feature>();
192
193         int numUnmatchedMatched=0;
194         int numMatchedMatched=0;
195         int numUnmatched=0;
196         int numMatched=0;
197
198         for (Feature ms2Feature : ms2Features.getFeatures())
199         {
200     if (ms2Feature.getDescription() != null && ms2Feature.getDescription
            ().contains("unmatched"))
201         numUnmatched++;
202     else
203         numMatched++;
204         }
205
206 //        Set<Feature> ms2MatchedSet = new HashSet<Feature>();
207
208         for (Feature ms1Feature : featureMatchingResult.
                getMasterSetFeatures())
209         {
210             matchedMs1FeatureHashSet.add(ms1Feature);
211
212             for (Feature ms2Feature : featureMatchingResult.get(
                    ms1Feature))
213             {
214                 matchedMs2FeatureHashSet.add(ms2Feature);
215
216                 String ms2Peptide = MS2ExtraInfoDef.getFirstPeptide(
                        ms2Feature);
217                 List<String> ms1PeptideList = MS2ExtraInfoDef.
                        getPeptideList(ms1Feature);
218
219                 if (ms1PeptideList != null && ms1PeptideList.contains(
                        ms2Peptide))
220                     continue;
221
222                 MS2ExtraInfoDef.addPeptideWithProtein(ms1Feature,
223                         ms2Peptide,
```

```
224                         MS2ExtraInfoDef.getFirstProtein(ms2Feature));
225                     //TODO: what the heck do I do about this?
226                     MS2ExtraInfoDef.setPeptideProphet(ms1Feature,
227                         MS2ExtraInfoDef.getPeptideProphet(ms2Feature));
228                     matchedPeptides.add(MS2ExtraInfoDef.getFirstPeptide(
                            ms2Feature));
229
230                 }
231 //if (featureMatchingResult.get(ms1Feature).size() > 1) System.err.
        println("Matched MS1 with " + featureMatchingResult.get(ms1Feature)
        .size() + " MS2 , list=" + MS2ExtraInfoDef.
        convertStringListToString(MS2ExtraInfoDef.getPeptideList(ms1Feature
        )));
232
233             }
234
235             ApplicationContext.infoMessage("Matched " +
                    matchedMs1FeatureHashSet.size() + " out of " + ms1Features.
                    getFeatures().length + " MS1 features.");
236             ApplicationContext.infoMessage("\t(" + matchedMs2FeatureHashSet.
                    size() + " out of " + ms2Features.getFeatures().length + "
                    MS2 features, (" + (matchedMs2FeatureHashSet.size()* 100 /
                    ms2Features.getFeatures().length) +"%))");
237
238             ApplicationContext.infoMessage("\t(" + matchedPeptides.size() +
239                     " distinct new peptides, out of " +
240                     allMS2Peptides.size() + ", " +
241                     (100 * matchedPeptides.size() / allMS2Peptides.size()) +
                            "%)");
242 //System.err.println("Self-matched matched: " + numMatchedMatched + " /
        " + numMatched + " ( " + 100 * (double)((double)numMatchedMatched /
         (double)numMatched) + "%)");
243 //System.err.println("Non-self-matched matched: " + numUnmatchedMatched
        + " / " + numUnmatched + " ( " + 100 * (double)((double)
        numUnmatchedMatched / (double)numUnmatched) + "%)");
244
245
246             double[] matchedIntensities = new double[
                    matchedMs1FeatureHashSet.size()];
247             int i=0;
248             for (Feature matchedMs1Feature : matchedMs1FeatureHashSet)
249                 matchedIntensities[i++] = matchedMs1Feature.getIntensity();
250             double meanMatchedIntensity = BasicStatistics.mean(
                    matchedIntensities);
251
252             System.err.println("Mean intensity of matched features: " +
                    meanMatchedIntensity);
253
254             Set<Feature> unmatchedMs1Features = new HashSet<Feature>();
255             for (Feature ms1Feature : ms1Features.getFeatures())
256                 if (!matchedMs1FeatureHashSet.contains(ms1Feature))
257                     unmatchedMs1Features.add(ms1Feature);
258             double[] unmatchedIntensities = new double[unmatchedMs1Features.
                    size()];
259             i=0;
260             for (Feature unmatchedMs1Feature : unmatchedMs1Features)
261                 unmatchedIntensities[i++] = unmatchedMs1Feature.getIntensity
                        ();
262             double meanUnmatchedIntensity = BasicStatistics.mean(
                    unmatchedIntensities);
263
264             System.err.println("Mean intensity of unmatched features: " +
                    meanUnmatchedIntensity);
265
266
267
268             if (writeUnmatched)
269             {
270                 for (Feature feature : ms1Features.getFeatures())
271                 {
272                     matchedMs1FeatureHashSet.add(feature);
273                 }
274             }
275
```

```
276             Feature[] annotatedFeatureArray = new Feature[
                    matchedMs1FeatureHashSet.size()];
277             int afaIndex = 0;
278             for (Feature feature : matchedMs1FeatureHashSet)
279                 annotatedFeatureArray[afaIndex++] = feature;
280
281             FeatureSet annotatedFeatureSet = new FeatureSet(
                    annotatedFeatureArray);
282             annotatedFeatureSet.addExtraInformationType(MS2ExtraInfoDef.
                    getSingletonInstance());
283
284             if (outFile != null)
285             {
286                 try
287                 {
288                     annotatedFeatureSet.save(outFile);
289                     ApplicationContext.infoMessage("Saved " +
                            annotatedFeatureSet.getFeatures().length +
290                                 " features, with annotations for matches, to
                                        file " + outFile.getAbsolutePath());
291                 }
292                 catch (Exception e)
293                 {
294                     throw new CommandLineModuleExecutionException(e);
295                 }
296             }
297
298             if (outUnmatchedMS2File != null)
299             {
300                 _log.debug("Writing unmatched MS2 features");
301                 List<Feature> unmatchedMs2FeatureList = new ArrayList<
                        Feature>();
302                 for (Feature ms2Feature : ms2Features.getFeatures())
303                 {
304                     if (!matchedPeptides.contains(MS2ExtraInfoDef.
                            getFirstPeptide(ms2Feature)))
305                         unmatchedMs2FeatureList.add(ms2Feature);
306                 }
307                 try
308                 {
309                     new FeatureSet(unmatchedMs2FeatureList.toArray(new
                            Feature[0])).save(outUnmatchedMS2File);
310                 }
311                 catch (IOException e)
312                 {
313                     throw new CommandLineModuleExecutionException(e);
314                 }
315             }
316
317             if (outAllMS2MarkedFile != null)
318             {
319                 _log.debug("Writing all MS2 features, marking unmatched with
                        description 'unmatched'");
320                 FeatureSet ms2FeaturesClone = (FeatureSet) ms2Features.clone
                        ();
321                 for (Feature ms2Feature : ms2FeaturesClone.getFeatures())
322                 {
323                     if (!matchedPeptides.contains(MS2ExtraInfoDef.
                            getFirstPeptide(ms2Feature)))
324                         ms2Feature.setDescription("unmatched");
325                 }
326                 try
327                 {
328                     ms2FeaturesClone.save(outAllMS2MarkedFile);
329                 }
330                 catch (IOException e)
331                 {
332                     throw new CommandLineModuleExecutionException(e);
333                 }
334             }
335
336             if (showCharts)
337             {
338                 List<Pair<Float, Float>> matchedFeatureTimes =
```

```
339                     new ArrayList<Pair<Float,Float>>();
340             for (Feature ms1Feature : featureMatchingResult.
                    getMasterSetFeatures())
341             {
342                 matchedMs1FeatureHashSet.add(ms1Feature);
343 
344                 for (Feature ms2Feature : featureMatchingResult.get(
                        ms1Feature))
345                 {
346                     matchedFeatureTimes.add(new Pair<Float,Float>(
                            ms1Feature.getTime(), ms2Feature.getTime()));
347                 }
348             }
349             float[][] scatterPlotData = new float[2][matchedFeatureTimes
                    .size()];
350             double[] histData = new double[matchedFeatureTimes.size()];
351 
352             for (int j=0; j<matchedFeatureTimes.size(); j++)
353             {
354                 Pair<Float,Float> pair = matchedFeatureTimes.get(j);
355                 scatterPlotData[0][j] = pair.first;
356                 scatterPlotData[1][j] = pair.second;
357 
358                 histData[j] = Math.abs(pair.first-pair.second);
359             }
360 
361             ScatterPlotDialog spd =
362                     new ScatterPlotDialog(scatterPlotData[0],
                            scatterPlotData[1], "");
363             spd.setAxisLabels("MS1 time","MS2 Time");
364             spd.setVisible(true);
365         }
366 
367 
368         //if there are peptides in the ms1 features, explore the
                agreement and disagreement
369         //with ms2 peptides
370         Set<String> ms1Peptides = new HashSet<String>();
371         for (Feature ms1Feature : ms1Features.getFeatures())
372             if (MS2ExtraInfoDef.getFirstPeptide(ms1Feature) != null)
373                 ms1Peptides.add(MS2ExtraInfoDef.getFirstPeptide(
                        ms1Feature));
374         if (ms1Peptides.size() > 0)
375         {
376             int numAgreement=0;
377             int numConflict=0;
378             int numAmbiguous=0;
379             for (Feature ms1Feature : featureMatchingResult.
                    getMasterSetFeatures())
380             {
381                 boolean ambiguous = false;
382                 List<Feature> matchedMS2Features = featureMatchingResult
                        .get(ms1Feature);
383                 Set<String> ms2PeptidesSet = new HashSet<String>();
384                 for (Feature feature : matchedMS2Features)
385                 {
386                     ms2PeptidesSet.add(MS2ExtraInfoDef.getFirstPeptide(
                            feature));
387                 }
388                 if (ms2PeptidesSet.size() > 1)
389                 {
390                     ambiguous=true;
391                 }
392                 List<String> ms2PeptideList = new ArrayList<String>();
393                 ms2PeptideList.addAll(ms2PeptidesSet);
394 
395                 List<String> ms1PeptideList = MS2ExtraInfoDef.
                        getPeptideList(ms1Feature);
396 
397                 if (ms1PeptideList == null)
398                     continue;
399 
400                 if (ms1PeptideList.size() > 1)
401                     ambiguous=true;
```

```
402
403                 Set<String> commonPeptides = new HashSet<String>();
404                 boolean agreement = false;
405
406                 for (String ms1Peptide : ms1PeptideList)
407                 {
408                     if (ms2PeptidesSet.contains(ms1Peptide))
409                         commonPeptides.add(ms1Peptide);
410                 }
411                 if (commonPeptides.size() > 0)
412                     agreement=true;
413
414                 if (ambiguous)
415                 {
416                     numAmbiguous++;
417                     ApplicationContext.infoMessage("Ambiguous:  MS1: " +
418                             MS2ExtraInfoDef.convertStringListToString(
                                    ms1PeptideList) + "\n\tamtmsw4 MS2: " +
419                             MS2ExtraInfoDef.convertStringListToString(
                                    ms2PeptideList));
420                 }
421                 if (agreement)
422                     numAgreement++;
423                 else
424                     numConflict++;
425             }
426             ApplicationContext.infoMessage("Peptide comparison, MS1 to
                    MS2:");
427             ApplicationContext.infoMessage("Agreement: " + numAgreement
                    + ", conflict: " +
428                     numConflict + ", ambiguous: " + numAmbiguous);
429         }
430     }
431
432     public class MyMatcher
433     {
434      public AmtFeatureSetMatcher.FeatureMatchingResult matchFeatures(
            FeatureSet ms1featureSet, FeatureSet ms2featureSet)
435      throws CommandLineModuleExecutionException
436      {
437       AmtFeatureSetMatcher.FeatureMatchingResult result = new
            AmtFeatureSetMatcher.FeatureMatchingResult();
438       Feature[] ms1features = ms1featureSet.getFeatures();
439       Feature[] ms2features =  ms2featureSet.getFeatures();
440     Arrays.sort(ms1features, scanMassAscComparator);
441     Arrays.sort(ms2features, scanMassAscComparator);
442
443     int ms1Index = 0;
444        for (Feature ms2feature : ms2features)
445        {
446         while (ms1features[ms1Index].getScan() < ms2feature.getScan())
447         {
448          ms1Index++;
449         }
450         if (ms1features[ms1Index].getScan() != ms2feature.getScan())
451         {
452          throw new CommandLineModuleExecutionException("ms1feature file
                and the ms2 pepXML file does not match up 100%.\n"+
453            "The mgf file mascot was searched with has to be created from
                the ms1feature file.\n"+
454            "Scans did not match.");
455         }
456         while (ms1features[ms1Index].getMass() < ms2feature.getMass())
457         {
458          ms1Index++;
459         }
460         if (ms1features[ms1Index].getMass() != ms2feature.getMass())
461         {
462          throw new CommandLineModuleExecutionException("ms1feature file
                and the ms2 pepXML file does not match up 100%.\n"+
463            "The mgf file mascot was searched with has to be created from
                the ms1feature file."+
464            "Masses did not match.");
465         }
```

```
466             result.add(ms1features[ms1Index], ms2feature);
467           }
468           return result;
469         }
470       }
471
472       public static final ScanMassAscComparator scanMassAscComparator =
              new ScanMassAscComparator();
473
474       static class ScanMassAscComparator implements Comparator<Feature>
475       {
476           public int compare(Feature o1, Feature o2)
477           {
478             if (o1.scan == o2.scan)
479             {
480               return _compareAsc(o1.mass, o2.mass);
481             }
482                 return _compareAsc(o1.scan, o2.scan);
483           }
484       }
485
486       static int _compareAsc(float a, float b)
487       {
488           return a == b ? 0 : a < b ? -1 : 1;
489       }
490   }
```

Listing E.6: Writes out a peptide array in a format easily analyzed in R

```
1   /*
2    * Copyright (c) 2003-2007 Fred Hutchinson Cancer Research Center
3    *
4    * Licensed under the Apache License, Version 2.0 (the "License");
5    * you may not use this file except in compliance with the License.
6    * You may obtain a copy of the License at
7    *
8    *      http://www.apache.org/licenses/LICENSE-2.0
9    *
10   * Unless required by applicable law or agreed to in writing, software
11   * distributed under the License is distributed on an "AS IS" BASIS,
12   * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
       implied.
13   * See the License for the specific language governing permissions and
14   * limitations under the License.
15   */
16  package org.fhcrc.cpl.viewer.commandline.modules;
17
18  import org.fhcrc.cpl.viewer.commandline.*;
19  import org.fhcrc.cpl.viewer.commandline.arguments.
        ArgumentValidationException;
20  import org.fhcrc.cpl.viewer.commandline.arguments.
        CommandLineArgumentDefinition;
21  import org.fhcrc.cpl.viewer.commandline.arguments.
        ArgumentDefinitionFactory;
22  import org.fhcrc.cpl.viewer.commandline.arguments.
        EnumeratedValuesArgumentDefinition;
23  import org.fhcrc.cpl.viewer.align.PeptideArrayAnalyzer;
24  import org.fhcrc.cpl.viewer.gui.util.ScatterPlotDialog;
25  import org.fhcrc.cpl.viewer.feature.Feature;
26  import org.fhcrc.cpl.viewer.feature.extraInfo.MS2ExtraInfoDef;
27  import org.labkey.common.tools.TabLoader;
28  import org.labkey.common.tools.ApplicationContext;
29  import org.labkey.common.util.Pair;
30  import org.apache.log4j.Logger;
31
32  import java.util.*;
33  import java.io.*;
34
35
36  public class MyPepArrayAnalyzer extends BaseCommandLineModuleImpl
37          implements CommandLineModule
38  {
39      protected static Logger _log = Logger.getLogger(
```

```
        PeptideArrayAnalyzerCommandLineModule.class);
40
41      protected File file;
42      protected File outFile;
43      protected File outDir;
44      protected File detailsFile;
45      boolean allowHalfMatched=false;
46      String[] caseRunNames;
47      String[] controlRunNames;
48      protected double minSignificantRatio = 3;
49
50      protected int minRunsForConsensusFeature = 2;
51
52      PeptideArrayAnalyzer peptideArrayAnalyzer = null;
53
54      protected boolean showCharts = false;
55
56      Object[] arrayRows;
57
58      protected int minPeptideSupport = 1;
59      protected int minFeatureSupport = 1;
60
61      protected static final int MODE_COMPARE_INTENSITIES_SAME_PEPTIDE =
            1;
62      protected static final int
            MODE_COMPARE_INTENSITIES_SAME_PEPTIDE_ADD_1 = 2;
63
64
65      protected final static String[] modeStrings =
66              {
67                      "comparepeptideintensities",
68                      "comparepeptideintensitiesadd1",
69              };
70
71      public static final String[] modeExplanations =
72              {
73                      "Compare intensity in the case runs vs. intensity in
                            the control runs",
74                      "Compare intensity in the case runs vs. intensity in
                            the control runs, adding 1, so that logs can
                            be used",
75              };
76
77      protected int mode=-1;
78
79  private boolean onlyMeanIntensities = false;
80
81      public MyPepArrayAnalyzer()
82      {
83          init();
84      }
85
86      protected void init()
87      {
88          mCommandName = "analyzemypeparray";
89
90          mShortDescription = "MINE! Tools for analyzing peptide arrays.";
91          mHelpMessage = "MINE! Tools for analyzing peptide arrays, for
                comparing MS2 results in one set of runs to "
92                  + "another and for summarizing overlap of MS1 features
                        found in different runs.";
93
94          CommandLineArgumentDefinition[] argDefs =
95                  {
96                      createEnumeratedArgumentDefinition("mode",true,
                            modeStrings, modeExplanations),
97                      createUnnamedArgumentDefinition(
                            ArgumentDefinitionFactory.FILE_TO_READ,true,
                            null),
98                      createFileToWriteArgumentDefinition("out",false, "
                            output file"),
99                      createFileToWriteArgumentDefinition("outdir",false,
                            "output directory"),
100
```

```
101                         createBooleanArgumentDefinition("allowhalfmatched",
                                false,"When␣determining␣whether␣peptide␣matches
                                ␣are␣made,␣should␣it␣be␣considered␣a␣match␣when
                                ␣one␣run␣has␣an␣ID␣and␣another␣run␣has␣an␣
                                intensity␣but␣no␣ID?",
102                                 allowHalfMatched),
103                         createFileToReadArgumentDefinition("caserunlistfile"
                                ,false,"File␣containing␣the␣names␣of␣runs␣in␣
                                the␣case␣group,␣one␣per␣line"),
104                         createFileToReadArgumentDefinition("
                                controlrunlistfile",false,"File␣containing␣the␣
                                names␣of␣runs␣in␣the␣control␣group,␣one␣per␣
                                line"),
105
106                         createDecimalArgumentDefinition("minsignificantratio
                                ",false,"Minimum␣ratio␣of␣intensities␣
                                considered␣interesting",
107                                 minSignificantRatio),
108                         createIntegerArgumentDefinition("
                                minconsensusfeatureruns",false,
109                                 "Minimum␣number␣of␣runs␣required␣for␣a␣
                                        feature␣to␣be␣included␣in␣the␣consensus
                                        ␣feature␣set",
110                                 minRunsForConsensusFeature),
111                         createIntegerArgumentDefinition("minpeptidesupport",
                                false,
112                                 "Minimum␣number␣of␣runs␣for␣which␣the␣same␣
                                        peptide␣was␣identified",
113                                 minPeptideSupport),
114                         createIntegerArgumentDefinition("minfeaturesupport",
                                false,
115                                 "Minimum␣number␣of␣runs␣for␣which␣a␣non-
                                        peptide-conflicting␣feature␣was␣
                                        identified",
116                                 minFeatureSupport),
117                            createBooleanArgumentDefinition("showcharts",
                                    false, "show␣charts?", showCharts),
118                 };
119         addArgumentDefinitions(argDefs);
120     }
121
122     public void assignArgumentValues()
123             throws ArgumentValidationException
124     {
125         mode = ((EnumeratedValuesArgumentDefinition)
                getArgumentDefinition("mode")).getIndexForArgumentValue(
                getStringArgumentValue("mode"));
126
127         file = getFileArgumentValue(CommandLineArgumentDefinition.
                UNNAMED_PARAMETER_VALUE_ARGUMENT);
128         outFile = getFileArgumentValue("out");
129         outDir = getFileArgumentValue("outdir");
130
131         showCharts = getBooleanArgumentValue("showcharts");
132
133         try
134         {
135             peptideArrayAnalyzer = new PeptideArrayAnalyzer(file);
136         }
137         catch (Exception e)
138         {
139             throw new ArgumentValidationException(e);
140         }
141
142         allowHalfMatched = getBooleanArgumentValue("allowhalfmatched");
143         File caseRunListFile = getFileArgumentValue("caserunlistfile");
144         File controlRunListFile = getFileArgumentValue("
                controlrunlistfile");
145
146         if (caseRunListFile != null)
147         {
148             try
149             {
150                 BufferedReader br = new BufferedReader(new FileReader(
```

```
                    caseRunListFile));
151                 List<String> caseRunNameList = new ArrayList<String>();
152                 while (true)
153                 {
154                     String line = br.readLine();
155                     if (null == line)
156                         break;
157                     if (line.length() == 0 || line.charAt(0) == '#')
158                         continue;
159                     caseRunNameList.add(line);
160                 }
161                 caseRunNames = caseRunNameList.toArray(new String[0]);
162 
163                 if (controlRunListFile != null)
164                 {
165                     br = new BufferedReader(new FileReader(
                            controlRunListFile));
166                     List<String> controlRunNameList = new ArrayList<
                            String>();
167                     while (true)
168                     {
169                         String line = br.readLine();
170                         if (null == line)
171                             break;
172                         if (line.length() == 0 || line.charAt(0) == '#')
173                             continue;
174                         controlRunNameList.add(line);
175                     }
176                     controlRunNames = controlRunNameList.toArray(new
                            String[0]);
177                 }
178                 else
179                 {
180                     List<String> controlRunNameList = new ArrayList<
                            String>();
181                     for (String runName : peptideArrayAnalyzer.
                            getRunNames())
182                         if (!caseRunNameList.contains(runName))
183                             controlRunNameList.add(runName);
184                     controlRunNames = controlRunNameList.toArray(new
                            String[controlRunNameList.size()]);
185                 }
186             }
187             catch (Exception e)
188             {
189                 throw new ArgumentValidationException(e);
190             }
191         }
192         else
193         {
194             List<String> runNames = peptideArrayAnalyzer.getRunNames();
195             if (runNames.size() == 2)
196             {
197                 ApplicationContext.setMessage("No␣case/control␣run␣names
                        ␣specified.␣␣Assuming␣run␣1␣is␣control,␣run␣2␣is␣
                        case");
198                 caseRunNames = new String[1];
199                 caseRunNames[0] = runNames.get(0);
200                 controlRunNames = new String[1];
201                 controlRunNames[0] = runNames.get(1);
202                 ApplicationContext.setMessage("Control␣run:␣" +
                        controlRunNames[0] + ",␣Case␣run:␣" + caseRunNames
                        [0]);
203             }
204         }
205 
206         peptideArrayAnalyzer.setCaseRunNames(caseRunNames);
207         peptideArrayAnalyzer.setControlRunNames(controlRunNames);
208 
209         minSignificantRatio = getDoubleArgumentValue("
                minsignificantratio");
210 
211         minRunsForConsensusFeature = getIntegerArgumentValue("
                minconsensusfeatureruns");
```

```
212
213             minPeptideSupport = getIntegerArgumentValue("minpeptidesupport")
                    ;
214             minFeatureSupport = getIntegerArgumentValue("minfeaturesupport")
                    ;
215         }
216
217
218         /**
219          * do the actual work
220          */
221         public void execute() throws CommandLineModuleExecutionException
222         {
223             try
224             {
225                 TabLoader tabLoader = new TabLoader(file);
226                 tabLoader.setReturnElementClass(HashMap.class);
227                 Object[] rows = tabLoader.load();
228                 ApplicationContext.setMessage("Array␣rows:␣" + rows.length);
229                 List<String> runNames = new ArrayList<String>();
230
231                 for (TabLoader.ColumnDescriptor column : tabLoader.
                        getColumns())
232                 {
233                     _log.debug("loading␣column␣" + column.name);
234                     if (column.name.startsWith("intensity_"))
235                     {
236                         runNames.add(column.name.substring("intensity_".
                                length()));
237                         _log.debug("adding␣run␣" + runNames.get(runNames.
                                size()-1));
238                     }
239                 }
240                 switch (mode)
241                 {
242                     case MODE_COMPARE_INTENSITIES_SAME_PEPTIDE:
243                         compareIntensitiesSamePeptide(rows, caseRunNames,
                                controlRunNames, false);
244                         break;
245                     case MODE_COMPARE_INTENSITIES_SAME_PEPTIDE_ADD_1:
246                         compareIntensitiesSamePeptide(rows, caseRunNames,
                                controlRunNames, true);
247                         break;
248                 }
249             }
250             catch (Exception e)
251             {
252                 throw new CommandLineModuleExecutionException(e);
253             }
254         }
255
256         /**
257          * Compare mean intensities of certain columns against each other in
                rows
258          * in which the same peptide is identified in enough runs.
259          *
260          * A lot of this is hardcoded for a particular purpose right now.
261          * @param rows
262          * @throws CommandLineModuleExecutionException
263          */
264
265         protected Pair<double[], double[]> compareIntensitiesSamePeptide(
                Object[] rows,
266                                                         String[] caseRunNames,
267                                                         String[]
                                                             controlRunNames,
268                                                         boolean add1)
269             throws CommandLineModuleExecutionException
270         {
271             if (caseRunNames == null || controlRunNames == null)
272                 throw new CommandLineModuleExecutionException("Error:␣You␣
                        must␣define␣case␣and␣control␣runs");
273
274             Set<String> peptidesHigherInCase =
```

```
275                 new HashSet<String>();
276             Set<String> peptidesHigherInControl =
277                 new HashSet<String>();
278             Set<String> peptidesHigherTotalInCase = new HashSet<String>();
279             Set<String> peptidesHigherTotalInControl = new HashSet<String>()
                    ;
280
281             int rowsHigherInControl = 0;
282             int rowsHigherInCase = 0;
283             int rowsHigherTotalInControl = 0;
284             int rowsHigherTotalInCase = 0;
285
286             Map<String, ProteinMapping> proteinMap = new HashMap<String,
                    ProteinMapping>();
287             // For the features without IDs
288             proteinMap.put(null, new ProteinMapping());
289             int numCases = 0, numControls = 0;
290
291             try
292             {
293                 int numPeptidesInAgreement = 0;
294                 int numPeptideConflicts=0;
295                 for (Object rowObj : rows)
296                 {
297                     HashMap rowMap = (HashMap) rowObj;
298
299                     List<Double[]> caseIntensities = new ArrayList<Double
                            []>(caseRunNames.length);
300                     List<Double[]> controlIntensities = new ArrayList<Double
                            []>(controlRunNames.length);
301                     int featureSupportCase = 0;
302                     int peptideSupportCase = 0;
303                     String peptide = null;
304                     String protein = null;
305                     double intensitySumCase = 0;
306                     double totalIntensitySumCase = 0;
307
308                     for (String caseRunName : caseRunNames)
309                     {
310                         try
311                         {
312                             Object runIntensity = rowMap.get("intensity_" +
                                    caseRunName);
313                             if (runIntensity == null && !add1)
314                                 continue;
315                          double intensity = Double.parseDouble(runIntensity.
                                toString());
316                             intensitySumCase += intensity;
317                             double totalIntensity = 0;
318                             try {
319                              runIntensity = rowMap.get("totalintensity_" +
                                    caseRunName);
320                              if (runIntensity == null && !add1)
321                                  continue;
322                           totalIntensity = Double.parseDouble(runIntensity.
                                toString());
323                              totalIntensitySumCase += totalIntensity;
324                             }
325                             catch (Exception e) {}
326                          caseIntensities.add(new Double[]{intensity,
                                totalIntensity});
327                         }
328                         catch (Exception e){}
329                         featureSupportCase++;
330                         String thisPeptide = (String) rowMap.get("peptide_"
                                + caseRunName);
331                         String thisProtein = (String) rowMap.get("protein_"
                                + caseRunName);
332                         if (thisPeptide == null && !add1)
333                             continue;
334
335
336                         if (add1)
337                         {
```

```
338                         if (peptide == null)
339                         {
340                             peptide = thisPeptide;
341                             assert thisProtein != null;
342                             protein = thisProtein;
343                         }
344                         if ((peptide != null && thisPeptide != null &&
345                             (!peptide.equals(thisPeptide))) ||
346                             (protein != null && thisProtein != null && !
                                    protein.equals(thisProtein)))
347                         {
348                             numPeptideConflicts++;
349                             peptideSupportCase = 0;
350                             break;
351                         }
352                     }
353                     else
354                     {
355                         if (peptide == null)
356                         {
357                             peptide = thisPeptide;
358                             assert thisProtein != null;
359                             protein = thisProtein;
360                             peptideSupportCase++;
361                         }
362                         else
363                         {
364                             try
365                             {
366                                 if (peptide.equals(thisPeptide) &&
                                        protein.equals(thisProtein))
367                                     peptideSupportCase++;
368                                 else
369                                 {
370                                     numPeptideConflicts++;
371                                     peptideSupportCase =0;
372                                     break;
373                                 }
374                             }
375                             catch (Exception e) {}
376                         }
377                     }
378                 }
379
380                 double intensityMeanCase = 0;
381                 double totalIntensityMeanCase = 0;
382                 if (featureSupportCase >= minFeatureSupport)
383                 {
384                         intensityMeanCase = intensitySumCase /
                                featureSupportCase;
385                         totalIntensityMeanCase = totalIntensitySumCase /
                                 featureSupportCase;
386                 }
387
388                 double intensitySumControl = 0;
389                 double totalIntensitySumControl = 0;
390                 int featureSupportControl = 0;
391                 int peptideSupportControl = 0;
392                 boolean peptideConflict = false;
393                 for (String controlRunName : controlRunNames)
394                 {
395                     try
396                     {
397                         Object runIntensity = rowMap.get("intensity_" +
                                controlRunName);
398                         if (runIntensity == null && !add1)
399                             continue;
400                          double intensity = Double.parseDouble(runIntensity.
                                 toString());
401                          intensitySumControl += intensity;
402                          double totalIntensity = 0;
403                          try
404                          {
405                              runIntensity = rowMap.get("totalintensity_" +
```

```
                                controlRunName);
406                         if (runIntensity == null && !add1)
407                             continue;
408                         totalIntensity = Double.parseDouble(runIntensity.
                                toString());
409                         totalIntensitySumControl += totalIntensity;
410                     }
411                     finally
412                     {
413                         controlIntensities.add(new Double[]{intensity,
                                totalIntensity});
414                     }
415                 }
416                 catch (Exception e){}
417
418                 featureSupportControl++;
419                 String thisPeptide = (String) rowMap.get("peptide_"
                        + controlRunName);
420                 String thisProtein = (String) rowMap.get("protein_"
                        + controlRunName);
421                 if (thisPeptide == null && !add1)
422                     continue;
423
424                 if (add1)
425                 {
426                     if (peptide == null)
427                     {
428                         peptide = thisPeptide;
429                         assert thisProtein != null;
430                         protein = thisProtein;
431                     }
432                     if ((peptide != null && thisPeptide != null &&
433                         !peptide.equals(thisPeptide)) ||
434                         (protein != null && thisProtein != null &&
435                         !protein.equals(thisProtein))
436                         )
437                     {
438                             numPeptideConflicts++;
439                             peptideSupportControl = 0;
440                             peptideConflict = true;
441                             break;
442                     }
443                 }
444                 else
445                 {
446                     if (peptide == null)
447                     {
448                         peptide = thisPeptide;
449                         assert thisProtein != null;
450                         protein = thisProtein;
451                         peptideSupportControl++;
452                     }
453                     else
454                     {
455                         if (peptide.equals(thisPeptide) && protein.
                                equals(thisProtein))
456                             peptideSupportControl++;
457                         else
458                         {
459                             numPeptideConflicts++;
460                             peptideSupportControl = 0;
461                             peptideConflict = true;
462                             break;
463                         }
464                     }
465                 }
466
467
468             }
469
470             double intensityMeanControl = 0;
471             double totalIntensityMeanControl = 0;
472             if (featureSupportControl >= minFeatureSupport)
473             {
```

```
474                         intensityMeanControl = intensitySumControl /
                                featureSupportControl;
475                         totalIntensityMeanControl = totalIntensitySumControl
                                / featureSupportControl;
476                     }
477
478     //if (peptide != null) System.err.println(peptide);
479
480                     boolean peptideAgreement = false;
481                     if (peptideSupportControl + peptideSupportCase >=
                            minPeptideSupport &&
482                             featureSupportCase >= minFeatureSupport &&
483                             featureSupportControl >= minFeatureSupport)
484                     {
485                         numPeptidesInAgreement++;
486                         peptideAgreement=true;
487                     }
488                     if (add1)
489                     {
490                         intensityMeanControl++;
491                         intensityMeanCase++;
492                         totalIntensityMeanControl++;
493                         totalIntensityMeanCase++;
494                     }
495
496                     if (peptide != null && (peptideAgreement || add1))
497                     {
498     //if (!peptideAgreement) System.err.println("no-agree peptide, peptide =
            " + peptide + ", add1= " + add1);
499                         double caseControlRatio = intensityMeanCase /
                                intensityMeanControl;
500                         if (caseControlRatio > minSignificantRatio)
501                         {
502                             rowsHigherInCase++;
503                             if (peptideAgreement)
504                                 peptidesHigherInCase.add(peptide);
505                         }
506                         else if (1 / caseControlRatio > minSignificantRatio)
507                         {
508                             rowsHigherInControl++;
509                             peptidesHigherInControl.add(peptide);
510                         }
511                         caseControlRatio = totalIntensityMeanCase /
                                totalIntensityMeanControl;
512                         if (caseControlRatio > minSignificantRatio)
513                         {
514                          rowsHigherTotalInCase++;
515                          if (peptideAgreement)
516                          {
517                           peptidesHigherTotalInCase.add(peptide);
518                          }
519                         }
520                         else
521                         {
522                          rowsHigherTotalInControl++;
523                          if (peptideAgreement)
524                          {
525                           peptidesHigherTotalInControl.add(peptide);
526                          }
527                         }
528                     }
529
530                     if (!peptideConflict && (peptide != null))
531                     {
532                         if (add1 || peptideAgreement)
533                         {
534     //if (!peptideAgreement) System.err.println("Adding one we wouldn't
            otherwise, add1 is " + add1 + ", mode is " + mode);
535                             Feature controlFeature = new Feature(1,1000,(
                                    float) intensityMeanControl);
536                             controlFeature.setTotalIntensity((float)
                                    totalIntensityMeanControl);
537                             MS2ExtraInfoDef.addPeptideWithProtein(
                                    controlFeature, peptide, protein);
```

```
538
539                             Feature caseFeature = new Feature(1,1000,(float)
                                    intensityMeanCase);
540                             caseFeature.setTotalIntensity((float)
                                    totalIntensityMeanCase);
541                             MS2ExtraInfoDef.addPeptideWithProtein(
                                    caseFeature, peptide, protein);
542
543                             ProteinMapping proteinFeatures = proteinMap.get(
                                    protein);
544                             if (proteinFeatures == null)
545                             {
546                              proteinFeatures = new ProteinMapping();
547                              proteinMap.put(protein, proteinFeatures);
548                             }
549                             proteinFeatures.add(caseFeature, caseIntensities
                                    , true);
550                             proteinFeatures.add(controlFeature,
                                    controlIntensities, false);
551                         }
552                     }
553                 }
554
555
556 //                  ApplicationContext.infoMessage("# Peptides in agreement: "
     + numPeptidesInAgreement);
557 //                  ApplicationContext.infoMessage("# Peptide conflicts: " +
    numPeptideConflicts);
558 //
559 //                  ApplicationContext.infoMessage("# Peptides higher in case:
     " + peptidesHigherInCase.size() + " peptides in " +
    rowsHigherInCase + " array rows");
560 //                  ApplicationContext.infoMessage("# Peptides higher in
    control: " + peptidesHigherInControl.size() + " peptides in " +
    rowsHigherInControl + " array rows");
561 //                  ApplicationContext.infoMessage("# Peptides higher in one
    or the other: " + (peptidesHigherInCase.size() +
    peptidesHigherInControl.size() +
562 //                                      " peptides in " + (rowsHigherInCase +
    rowsHigherInControl) + " array rows"));
563 //                  ApplicationContext.infoMessage("# Peptides higher (
    totalintensity) in case: " + peptidesHigherInCase.size() + "
    peptides in " + rowsHigherTotalInCase + " array rows");
564 //                  ApplicationContext.infoMessage("# Peptides higher (
    totalintensity) in control: " + peptidesHigherInControl.size() + "
    peptides in " + rowsHigherTotalInControl + " array rows");
565
566
567                     double maxIntensity = 0d;
568                     int numInsideTwofold = 0;
569
570                 for (Map.Entry<String, ProteinMapping> mapping : proteinMap.
                        entrySet())
571                 {
572                  ProteinMapping protMapping = mapping.getValue();
573                  if (protMapping.maxIntensity() > maxIntensity)
574                  {
575                   maxIntensity = protMapping.maxIntensity();
576                  }
577                  numInsideTwofold += protMapping.numInsideTwofold();
578                  numCases += protMapping.caseFeatures.size();
579                  numControls += protMapping.controlFeatures.size();
580                 }
581
582                     if (showCharts)
583                     {
584                         System.err.println("protein groups on plot: " +
                                proteinMap.size());
585                         ScatterPlotDialog spd = new ScatterPlotDialog();
586                         ScatterPlotDialog spd2 = new ScatterPlotDialog();
587                         for (Map.Entry<String, ProteinMapping> mapping :
                                proteinMap.entrySet())
588                         {
589                          ProteinMapping protMapping = mapping.getValue();
```

```
590                     spd.addData(protMapping.getControlIntensities(),
                            protMapping.getCaseIntensities(),
591                       "intensities: x is control, y is case, protein "+
                              mapping.getKey());
592                     spd2.addData(protMapping.getControlLogIntensities(),
                            protMapping.getCaseLogIntensities(),
593                       "intensities: x is log control, y is log case, 
                              protein "+mapping.getKey());
594                   }
595
596     double[] line = new double[(int) maxIntensity+1];
597     double[] logLine = new double[(int) maxIntensity+1];
598     for (int c=1; c<line.length; c++)
599     {
600         line[c] = c;
601         logLine[c] = Math.log(c);
602     }
603     spd.addData(line, line, "1:1 line");
604     spd2.addData(logLine, logLine, "1:1 line");
605
606                 spd.setVisible(true);
607                 spd2.setVisible(true);
608             }
609             System.err.println("Same peptide intensity summary:");
610 System.err.println("Within twofold: " + numInsideTwofold + " out of " +
      numCases);
611
612             if (outFile != null)
613             {
614                 ApplicationContext.infoMessage("Writing ratios to file "
                        + outFile.getAbsolutePath());
615                 PrintWriter outPW = null;
616                 try
617                 {
618
619                     outPW = new PrintWriter(outFile);
620                     if (onlyMeanIntensities)
621                     {
622                      outPW.println("protein\tpeptide\tratio\tcase\
                            tcontrol");
623                     }
624                     else
625                     {
626                      outPW.println("protein\tpeptide\tintensity\
                            ttotalintensity\tcaseorcontrol");
627                     }
628                     for (Map.Entry<String, ProteinMapping> mapping :
                          proteinMap.entrySet())
629                     {
630                      ProteinMapping protMapping = mapping.getValue();
631                      String protein = mapping.getKey();
632                         for (int i=0; i<protMapping.caseFeatures.size();
                                i++)
633                         {
634                          Feature caseFeature = protMapping.caseFeatures.
                                get(i);
635                          Feature controlFeature = protMapping.
                                controlFeatures.get(i);
636                             String peptide = MS2ExtraInfoDef.
                                   getFirstPeptide(caseFeature);
637                             if (onlyMeanIntensities)
638                             {
639                              outPW.println(protein + "\t" + peptide + "\
                                    t" + (caseFeature.getIntensity() /
                                    controlFeature.getIntensity()) + "\t"
                                    + caseFeature.getIntensity() + "\t" +
                                    controlFeature.getIntensity());
640                             }
641                             else
642                             {
643                              for (Double[] intensity : protMapping.
                                    caseIntensities.get(i))
644                              {
645                               outPW.println(protein + "\t" + peptide + "
```

```
                                        \t" + intensity[0] + "\t" + intensity
                                        [1] + "\tcase");
646                                 }
647                                 for (Double[] intensity : protMapping.
                                        controlIntensities.get(i))
648                                 {
649                                  outPW.println(protein + "\t" + peptide + "
                                        \t" + intensity[0] + "\t" + intensity
                                        [1] + "\tcontrol");
650                                 }
651                                }
652                            }
653                        }
654                    }
655                    catch (Exception e)
656                    {
657                        throw new CommandLineModuleExecutionException(e);
658                    }
659                    finally
660                    {
661                        if (outPW != null)
662                            outPW.close();
663                    }
664                    ApplicationContext.infoMessage("Done writing ratios");
665                }
666            }
667            catch (Exception e)
668            {
669                throw new CommandLineModuleExecutionException(e);
670            }
671            double[] intensitiesCasearray = new double[numCases];
672            double[] intensitiesControlarray = new double[numControls];
673            int x = 0;
674            for (Map.Entry<String, ProteinMapping> mapping : proteinMap.
                   entrySet())
675            {
676             ProteinMapping protMapping = mapping.getValue();
677                for (int i=0; i<protMapping.caseFeatures.size(); i++)
678                {
679                 Feature caseFeature = protMapping.caseFeatures.get(i);
680                 Feature controlFeature = protMapping.controlFeatures.get(i)
                        ;
681                 intensitiesCasearray[x] = caseFeature.getIntensity();
682                 intensitiesControlarray[x] = controlFeature.getIntensity();
683                 x++;
684                }
685            }
686            return new Pair<double[], double[]>(intensitiesCasearray,
                   intensitiesControlarray);
687       }
688
689       class ProteinMapping
690       {
691        private List<List<Double[]>> controlIntensities = new ArrayList<
              List<Double[]>>();
692     private List<List<Double[]>> caseIntensities = new ArrayList<List<
           Double[]>>();
693     private List<Feature> caseFeatures = new ArrayList<Feature>();
694        private List<Feature> controlFeatures = new ArrayList<Feature>();
695        private double maxIntensity = Double.NaN;
696        private int numInsideTwofold = 0;
697        private boolean calculated = false;
698
699        double maxIntensity() {
700         if (!calculated)
701         {
702          calc();
703         }
704      return maxIntensity;
705       }
706
707       public void add(Feature feature, List<Double[]> caseIntensities2,
              boolean isCase)
708     {
```

```
709         if (isCase)
710         {
711          caseFeatures.add(feature);
712          caseIntensities.add(caseIntensities2);
713         }
714         else
715         {
716          controlFeatures.add(feature);
717          controlIntensities.add(caseIntensities2);
718         }
719         calculated = false;
720     }
721
722     int numInsideTwofold()
723        {
724         if (!calculated)
725         {
726          calc();
727         }
728         return numInsideTwofold;
729        }
730
731        void calc()
732        {
733         maxIntensity = Double.NEGATIVE_INFINITY;
734         numInsideTwofold = 0;
735               for (int i = 0; i < caseFeatures.size(); i++)
736           {
737               Feature caseFeature = caseFeatures.get(i);
738               Feature controlFeature = controlFeatures.get(i);
739                  if (caseFeature.getIntensity() > maxIntensity) {
740                   maxIntensity = caseFeature.getIntensity();
741                  }
742                  if (controlFeature.getIntensity() > maxIntensity) {
743                   maxIntensity = controlFeature.getIntensity();
744                  }
745                  double intensitiesRatio = caseFeature.getIntensity() /
                         controlFeature.getIntensity();
746
747                  if (intensitiesRatio > 0.5 && intensitiesRatio < 2.0)
748                      numInsideTwofold++;
749           }
750              calculated = true;
751        }
752
753        double[] getControlIntensities()
754        {
755         double[] intenList = new double[controlFeatures.size()];
756         for (int i = 0; i < intenList.length; i++)
757         {
758          Feature feature = controlFeatures.get(i);
759             intenList[i] = feature.getIntensity();
760         }
761         return intenList;
762        }
763
764        double[] getCaseIntensities()
765        {
766         double[] intenList = new double[caseFeatures.size()];
767         for (int i = 0; i < intenList.length; i++)
768         {
769          Feature feature = caseFeatures.get(i);
770             intenList[i] = feature.getIntensity();
771         }
772         return intenList;
773        }
774
775        double[] getControlLogIntensities()
776        {
777         double[] intenList = new double[controlFeatures.size()];
778         for (int i = 0; i < intenList.length; i++)
779         {
780          Feature feature = controlFeatures.get(i);
781             intenList[i] = Math.log(feature.getIntensity());
```

```
782         }
783         return intenList;
784        }
785
786        double[] getCaseLogIntensities()
787        {
788         double[] intenList = new double[caseFeatures.size()];
789         for (int i = 0; i < intenList.length; i++)
790         {
791          Feature feature = caseFeatures.get(i);
792             intenList[i] = Math.log(feature.getIntensity());
793         }
794         return intenList;
795        }
796       }
797 }
```

Listing E.7: Retrieves Mascot XML files and outputs details easily analyzed in R

```
1  package org.fhcrc.cpl.viewer.commandline.modules;
2
3  import java.io.BufferedInputStream;
4  import java.io.BufferedReader;
5  import java.io.File;
6  import java.io.FileWriter;
7  import java.io.IOException;
8  import java.io.InputStream;
9  import java.io.InputStreamReader;
10 import java.io.PrintWriter;
11 import java.io.UnsupportedEncodingException;
12 import java.net.MalformedURLException;
13 import java.net.URL;
14 import java.net.URLEncoder;
15 import java.util.Enumeration;
16 import java.util.List;
17 import java.util.Properties;
18
19 import org.apache.log4j.Logger;
20 import org.fhcrc.cpl.viewer.commandline.
       CommandLineModuleExecutionException;
21 import org.fhcrc.cpl.viewer.commandline.arguments.
       ArgumentValidationException;
22 import org.fhcrc.cpl.viewer.commandline.arguments.
       CommandLineArgumentDefinition;
23 import org.fhcrc.cpl.viewer.feature.filehandler.
       MascotXMLFeatureFileHandler;
24 import org.fhcrc.cpl.viewer.feature.filehandler.
       MascotXMLFeatureFileHandler.ProteinHit;
25
26 public class MascotXMLToTsvCLM extends BaseCommandLineModuleImpl
27 {
28  private static final String TAB_SEP = "\t";
29
30  protected static Logger _log = Logger
31    .getLogger(ExtractRunsFromPepXmlCommandLineModule.class);
32
33  protected File inMascotXmlFile = null;
34  protected File outDir = null;
35  protected String sourceFileName = null;
36
37  protected static final int FILE_FORMAT_TSV = 0;
38  protected static final int FILE_FORMAT_PEPXML = 1;
39
40  protected int outFormat = 1;
41
42  protected boolean populateTimes = false;
43  protected File mzXmlDir = null;
44
45  protected String _url = "http://mascot4.bmb.sdu.dk/mascot/cgi";
46
47  @SuppressWarnings("unused")
48  private int errorCode;
```

```
49
50  private String errorString;
51
52  private String _proxyURL;
53
54  protected static final String[] formatStrings = { "tsv", "pepxml" };
55
56  public MascotXMLToTsvCLM()
57  {
58   init();
59  }
60
61  protected void init()
62  {
63   mCommandName = "mascotxmltotsvclm";
64   mShortDescription = "Make a tsv file from a Mascot XML file.";
65   mHelpMessage = "Parses a Mascot XML file in order to generate a tsv
         file with protein summary data.";
66
67   CommandLineArgumentDefinition[] argDefs = {
68     createFileToReadArgumentDefinition(
69       CommandLineArgumentDefinition.UNNAMED_PARAMETER_VALUE_ARGUMENT,
70       true, "Input MascotXML file"),
71     createDirectoryToReadArgumentDefinition("outdir", true,
72       "Output Folder"),
73     createStringArgumentDefinition("mascotURL", true,
74       "ie. http://mascot.server.com/mascot/cgi", _url),
75   // createEnumeratedArgumentDefinition("outformat", false, "Output
76   // format", formatStrings, "pepxml"),
77   // createBooleanArgumentDefinition("populatetimes", false, "Populate
78   // times using mzXML file", populateTimes),
79   // createDirectoryToReadArgumentDefinition("mzxmldir", false, "
         Directory
80   // to search for mzXML files (for populating times)")
81   };
82   addArgumentDefinitions(argDefs);
83  }
84
85  public void assignArgumentValues() throws ArgumentValidationException
86  {
87   inMascotXmlFile = getFileArgumentValue(CommandLineArgumentDefinition.
         UNNAMED_PARAMETER_VALUE_ARGUMENT);
88
89   outDir = getFileArgumentValue("outdir");
90   if (!outDir.isDirectory())
91   {
92    throw new ArgumentValidationException(outDir.getAbsolutePath()
93      + " is not a directory.");
94   }
95
96   // String outFormatString = getStringArgumentValue("outformat");
97   // for (int i=0; i<formatStrings.length; i++)
98   // {
99   // String formatString = formatStrings[i];
100  // if (formatString.equalsIgnoreCase(outFormatString))
101  // {
102  // outFormat = i;
103  // break;
104  // }
105  // }
106
107  // populateTimes = getBooleanArgumentValue("populatetimes");
108  // if (populateTimes)
109  // {
110  // assertArgumentPresent("mzxmldir","populatetimes");
111  // mzXmlDir = getFileArgumentValue("mzxmldir");
112  // }
113  }
114
115  /**
116   * do the actual work
117   */
118  public void execute() throws CommandLineModuleExecutionException
119  {
```

```
120   try
121   {
122    String[][] filesToGet = {
123      { "iain mpds mix1 (msInspect processing2)",
124        "../data/20080604/F053281.dat",
125        "../data/20080604/F053282.dat",
126        "../data/20080604/F053283.dat", },
127      { "iain mpds mix2 (msInspect processing2)",
128        "../data/20080604/F053297.dat",
129        "../data/20080604/F053298.dat",
130        "../data/20080604/F053299.dat", },
131      { "iain mpds mix1 (msInspect processing)",
132        "../data/20080530/F052866.dat",
133        "../data/20080530/F052867.dat",
134        "../data/20080530/F052868.dat", },
135      { "iain mpds mix1 (msInspect processing) (40ppm)",
136        "../data/20080603/F053098.dat",
137        "../data/20080603/F053101.dat",
138        "../data/20080603/F053102.dat", },
139      { "iain mpds mix1 (plgs processing)",
140        "../data/20080522/F052044.dat",
141        "../data/20080522/F051989.dat",
142        "../data/20080522/F051987.dat", },
143      { "iain mpds mix2 (msInspect processing)",
144        "../data/20080530/F052869.dat",
145        "../data/20080530/F052870.dat",
146        "../data/20080530/F052871.dat", },
147  //       { "iain mpds mix1 +ecoli (msInspect) merged",
148  //         "../data/20080603/F053105.dat"
149  //       },
150  //       { "iain mpds mix1 +ecoli (msInspect)",
151  //         "../data/20080603/F053104.dat",
152  //         "../data/20080603/F053106.dat",
153  //         "../data/20080603/F053107.dat"
154  //       },
155      { "hye mpds mix1 30sec (plgs processing)",
156        "../data/20080602/F053022.dat",
157        "../data/20080602/F053023.dat",
158        "../data/20080602/F053024.dat", },
159      { "hye mpds mix1 2sec (plgs processing)",
160        "../data/20080602/F053025.dat",
161        "../data/20080602/F053026.dat",
162        "../data/20080602/F053027.dat", },
163      { "hye mpds mix1 mse (plgs processing)",
164        "../data/20080602/F053028.dat",
165        "../data/20080602/F053029.dat",
166        "../data/20080602/F053030.dat", },
167      { "hye mpds mix1 mse (msInspect processing)",
168        "../data/20080603/F053063.dat",
169        "../data/20080603/F053064.dat",
170        "../data/20080603/F053065.dat", },
171      { "iain mpds mix1+ecoli (plgs)",
172        "../data/20080610/F053760.dat",
173        "../data/20080610/F053761.dat",
174        "../data/20080610/F053762.dat", },
175    };
176    String suffix = ".protein.tsv";
177
178    MascotXMLFeatureFileHandler mascotFileHandler =
           MascotXMLFeatureFileHandler
179      .getSingletonInstance();
180    File newSaveFile = new File(outDir.getPath() + File.separatorChar
181      + "alle_samlet" + suffix);
182    PrintWriter pw = new PrintWriter(newSaveFile);
183    pw.println("method" + TAB_SEP + "method.nr" + TAB_SEP
184      + "repetition" + TAB_SEP + "Coverage" + TAB_SEP + "Matches"
185      + TAB_SEP + "Peptide RMS" + TAB_SEP + "Fragments" + TAB_SEP
186      + "Accession" + TAB_SEP + "Score" + TAB_SEP + "Expect"
187      + TAB_SEP + "Description");
188    for (int fg = 0; fg < filesToGet.length; fg++)
189    {
190     String[] groupToGet = filesToGet[fg];
191     for (int i = 1; i < groupToGet.length; i++)
192     {
```

```
193         File fileToGet = new File(groupToGet[i]);
194         File fileToSave = new File(outDir.getPath()
195           + File.separatorChar + fileToGet.getName());
196         if (fileToSave.exists())
197         {
198          _log.debug("skipping existing file: " + groupToGet[i]);
199         } else
200         {
201          _log.debug("get file: " + groupToGet[i]);
202          if (!getFile(groupToGet[i], fileToSave))
203          {
204           throw new Exception("ERROR");
205          }
206         }
207         List<ProteinHit> proteinHits = mascotFileHandler
208           .loadProteinHits(fileToSave);
209
210         for (ProteinHit ph : proteinHits)
211         {
212          if (ph.getAcc().startsWith("P00330") ||
213            ph.getAcc().startsWith("P00924") ||
214            ph.getAcc().startsWith("P02769") ||
215            ph.getAcc().startsWith("P00489"))
216          {
217           pw.println(groupToGet[0] + TAB_SEP + "n"+(fg + 1) + TAB_SEP
218            + i + TAB_SEP + ph.getCoverage() + TAB_SEP
219            + ph.getMatches() + TAB_SEP + ph.getRms()
220            + TAB_SEP + ph.getPepFragments() + TAB_SEP
221            + ph.getAcc().split("[ |]")[0] + TAB_SEP
222            + ph.getScore() + TAB_SEP + ph.getExpect()
223            + TAB_SEP + ph.getDesc());
224          }
225         }
226        }
227       }
228       pw.close();
229      } catch (Exception e)
230      {
231       throw new CommandLineModuleExecutionException("Error!", e);
232      }
233     }
234
235     protected boolean getFile(String fileToGet, File fileToSave)
236       throws IOException
237     {
238      errorCode = 0;
239      errorString = "";
240
241      if ("".equals(fileToGet) || "".equals(fileToSave))
242      {
243       _log.error("At least one of the required arguments is empty.");
244       return false;
245      }
246
247      // get ./export_dat_2.pl
248
249      String[][] submitFields = {
250        { "cgi", "export_dat_2.pl" },
251        { "file", fileToGet },
252        { "do_export", "1" },
253        { "prot_hit_num", "1" },
254        { "prot_acc", "1" },
255        { "pep_query", "1" },
256        { "pep_rank", "1" },
257        { "pep_isbold", "1" },
258        { "pep_exp_mz", "1" },
259        { "_showallfromerrortolerant", "" },
260        { "_onlyerrortolerant", "" },
261        { "_noerrortolerant", "" },
262        { "_show_decoy_report", "" },
263        { "export_format", "XML" },
264        { "_sigthreshold", "0.05" },
265        { "REPORT", "AUTO" },
266        // _server_mudpit_switch standard=99999999 mudpit=0.000000001
```

```
267       { "_server_mudpit_switch", "0.000000001" },
268       { "_ignoreionsscorebelow", "15" },
269       // {show_same_sets
270       { "_showsubsets", "0" },
271       // {_requireboldred
272       { "search_master", "1" },
273       { "show_header", "1" },
274       { "show_mods", "1" },
275       { "show_params", "1" },
276       { "show_format", "1" },
277       // {show_masses
278       { "protein_master", "1" }, { "prot_score", "1" },
279       { "prot_desc", "1" }, { "prot_mass", "1" },
280       { "prot_matches", "1" }, { "prot_cover", "1" },
281       { "prot_len", "1" }, { "prot_pi", "1" },
282       { "prot_tax_str", "1" }, { "prot_tax_id", "1" },
283       { "prot_seq", "1" }, { "prot_empai", "1" },
284       { "peptide_master", "1" }, { "pep_exp_mr", "1" },
285       { "pep_exp_z", "1" }, { "pep_calc_mr", "1" },
286       { "pep_delta", "1" }, { "pep_start", "1" }, { "pep_end", "1" },
287       { "pep_miss", "1" }, { "pep_score", "1" },
288       { "pep_homol", "1" }, { "pep_ident", "1" },
289       { "pep_expect", "1" }, { "pep_seq", "1" },
290       // {pep_frame
291       { "pep_var_mod", "1" }, { "pep_num_match", "1" },
292       { "pep_scan_title", "1" }, { "show_unassigned", "1" },
293     // {query_master
294     // {query_title
295     // {query_qualifiers
296     // {query_params
297     // {query_peaks
298     // {query_raw
299     };
300     Properties parameters = new Properties();
301     for (String[] submitField : submitFields)
302     {
303      parameters.setProperty(submitField[0], submitField[1]);
304     }
305
306     Properties results = request(parameters, false);
307     if (!results.getProperty("error", "0").equals("0"))
308     {
309      throw new IOException(results.getProperty("errorstring",
310        "I don't know what went wrong!"));
311     }
312
313     FileWriter fileWriter = new FileWriter(fileToSave);
314     fileWriter.write(results.getProperty("HTTPContent"));
315     fileWriter.close();
316     return true;
317   }
318
319   private String requestURL(Properties parameters)
320   {
321    StringBuffer requestURLLSB = new StringBuffer(_url);
322    if (!_url.endsWith("/"))
323    {
324     requestURLLSB.append("/");
325    }
326    requestURLLSB.append(parameters.getProperty("cgi", "login.pl"));
327    requestURLLSB.append("?");
328    boolean firstEntry = true;
329    for (Enumeration e = parameters.propertyNames(); e.hasMoreElements();)
330    {
331     String s = (String) e.nextElement();
332     if (!"cgi".equalsIgnoreCase(s))
333     {
334      if (firstEntry)
335      {
336       firstEntry = false;
337      } else
338      {
339       requestURLLSB.append("&");
340      }
```

```
341       try
342       {
343        requestURLLSB.append(URLEncoder.encode(s, "UTF-8"));
344       } catch (UnsupportedEncodingException x)
345       {
346        requestURLLSB.append(s);
347       }
348       String val = parameters.getProperty(s);
349       if (!"".equals(val))
350       {
351        requestURLLSB.append("=");
352        try
353        {
354         requestURLLSB.append(URLEncoder.encode(val, "UTF-8"));
355        } catch (UnsupportedEncodingException x)
356        {
357         requestURLLSB.append(val);
358        }
359       }
360      }
361     }
362
363     return requestURLLSB.toString();
364    }
365
366    private Properties request(Properties parameters, boolean parse)
367    {
368     // connect to the Mascot Server to send request
369     // report the results as a property set, i.e. key=value pairs
370
371     Properties results = new Properties();
372     String mascotRequestURL = requestURL(parameters);
373     try
374     {
375      URL mascotURL = new URL(mascotRequestURL);
376      if (parse)
377      {
378       InputStream in = new BufferedInputStream(mascotURL.openStream());
379       results.load(in);
380       in.close();
381       errorString = results.getProperty("errorstring", "");
382      } else
383      {
384       BufferedReader in = new BufferedReader(new InputStreamReader(
385         mascotURL.openStream()));
386       String str;
387       StringBuffer reply = new StringBuffer();
388       while ((str = in.readLine()) != null)
389       {
390        reply.append(str);
391        reply.append("\n");
392       }
393       results.setProperty("HTTPContent", reply.toString());
394       in.close();
395      }
396     } catch (MalformedURLException x)
397     {
398      String password = parameters.getProperty("password", "");
399      if (password.length() > 0)
400       mascotRequestURL = mascotRequestURL.replace(password, "***");
401      // If using the class logger, then assume user interface will
402      // deliver the error message.
403      String msg = "Exception "
404        + x.getClass()
405        + " connect("
406        + _url
407        + ","
408        + parameters.getProperty("username", "<null>")
409        + ","
410        + (parameters.getProperty("password", "").length() > 0 ? "***"
411          : "") + "," + _proxyURL + ")=" + mascotRequestURL;
412      _log.debug(msg);
413      errorCode = 1;
414      errorString = "Fail to parse Mascot Server URL";
```

```
415     results.setProperty("error", "1");
416     results.setProperty("errorstring", errorString);
417     results.setProperty("exceptionmessage", x.getMessage());
418     results.setProperty("exceptionclass", x.getClass().getName());
419    } catch (Exception x)
420    {
421     String password = parameters.getProperty("password", "");
422     if (password.length() > 0)
423      mascotRequestURL = mascotRequestURL.replace(password, "***");
424     // If using the class logger, then assume user interface will
425     // deliver the error message.
426     String msg = "Exception " + x.getClass() + " connect(" + _url + ","
427       + parameters.getProperty("username", "<null>") + ","
428       + (password.length() > 0 ? "***" : "") + "," + _proxyURL
429       + ")=" + mascotRequestURL;
430     _log.debug(msg);
431     errorCode = 2;
432     errorString = "Fail to interact with Mascot Server";
433     results.setProperty("error", "2");
434     results.setProperty("errorstring", errorString);
435     results.setProperty("exceptionmessage", x.getMessage());
436     results.setProperty("exceptionclass", x.getClass().getName());
437    }
438
439    return results;
440   }
441  }
```

Listing E.8: Reads a Mascot XML file and returns a featureset

```
1   /*
2    * Copyright (c) 2003-2007 Fred Hutchinson Cancer Research Center
3    *
4    * Licensed under the Apache License, Version 2.0 (the "License");
5    * you may not use this file except in compliance with the License.
6    * You may obtain a copy of the License at
7    *
8    *      http://www.apache.org/licenses/LICENSE-2.0
9    *
10   * Unless required by applicable law or agreed to in writing, software
11   * distributed under the License is distributed on an "AS IS" BASIS,
12   * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
         implied.
13   * See the License for the specific language governing permissions and
14   * limitations under the License.
15   */
16  package org.fhcrc.cpl.viewer.feature.filehandler;
17
18  import org.fhcrc.cpl.viewer.feature.FeatureSet;
19  import org.fhcrc.cpl.viewer.feature.Feature;
20  import org.fhcrc.cpl.viewer.feature.FeaturePepXmlWriter;
21  import org.fhcrc.cpl.viewer.feature.extraInfo.MS2ExtraInfoDef;
22  import org.apache.log4j.Logger;
23  import org.apache.xmlbeans.XmlException;
24  import org.labkey.common.tools.*;
25
26  import com.matrixscience.xmlns.schema.mascotSearchResults2.
        MascotSearchResultsDocument;
27  import com.matrixscience.xmlns.schema.mascotSearchResults2.PeptideType;
28  import com.matrixscience.xmlns.schema.mascotSearchResults2.
        MascotSearchResultsDocument.MascotSearchResults;
29  import com.matrixscience.xmlns.schema.mascotSearchResults2.
        MascotSearchResultsDocument.MascotSearchResults.Header;
30  import com.matrixscience.xmlns.schema.mascotSearchResults2.
        MascotSearchResultsDocument.MascotSearchResults.SearchParameters;
31  import com.matrixscience.xmlns.schema.mascotSearchResults2.
        MascotSearchResultsDocument.MascotSearchResults.Hits.Hit;
32  import com.matrixscience.xmlns.schema.mascotSearchResults2.
        MascotSearchResultsDocument.MascotSearchResults.Hits.Hit.Protein;
33
34  import javax.xml.stream.XMLStreamException;
35  import java.io.*;
36  import java.math.BigInteger;
```

```
37  import java.util.ArrayList;
38  import java.util.HashMap;
39  import java.util.List;
40  import java.util.Map;
41
42  /**
43   * File handler for native msInspect feature files
44   */
45  public class MascotXMLFeatureFileHandler extends
        BaseFeatureSetFileHandler
46  {
47   static Logger _log = Logger.getLogger(MascotXMLFeatureFileHandler.class
         );
48
49   protected int firstSpectrumQueryIndex = 1;
50
51   public static final String FILE_TYPE_NAME = "MASCOTXML";
52
53   private List<ProteinHit> proteinHits = new ArrayList<ProteinHit>();
54
55   // / The score names used by Mascot when it generates pepxml files.
56   public static final String IONSCORE = "ionscore";
57   public static final String IDENTITYSCORE = "identityscore";
58   public static final String EXPECT = "expect";
59
60   // XXX Should really be handled by the TabLoader. TabLoader needs to be
         able
61   // to take a string at a time and parse it
62   public static final String SCANTITLE_SEP_CHAR = "\t";
63   public static final int SCANTITLE_SCAN = 0;
64   public static final int SCANTITLE_TIME = 1;
65   public static final int SCANTITLE_INTENSITY = 5;
66
67   protected static MascotXMLFeatureFileHandler singletonInstance = null;
68
69   public static MascotXMLFeatureFileHandler getSingletonInstance()
70   {
71    if (singletonInstance == null)
72     singletonInstance = new MascotXMLFeatureFileHandler();
73    return singletonInstance;
74   }
75
76   /**
77    * Loads the Mascot XML file into a List of ProteinHit.
78    */
79   public List<ProteinHit> loadProteinHits(File file) throws IOException
80   {
81    try
82    {
83     proteinHits.clear();
84     MascotSearchResultsDocument msrdoc = MascotSearchResultsDocument.
           Factory
85       .parse(file);
86     MascotSearchResults msr = msrdoc.getMascotSearchResults();
87     // for (Modification msVarMod :
88     // msr.getVariableMods().getModificationArray())
89     // {
90     // String name = msVarMod.getName();
91     // double delta = msVarMod.getDelta();
92     // MS2Modification ms2mod = new MS2Modification();
93     // ms2mod.setMassDiff((float) delta);
94     // }
95     // Header msHeader = msr.getHeader();
96
97     // / Reused in the inner loop
98     Map<String, String> scoreMap = new HashMap<String, String>();
99     for (Hit msHit : msr.getHits().getHitArray())
100    {
101     Protein[] msProteinArr = msHit.getProteinArray();
102     for (Protein msProtein : msProteinArr)
103     {
104      ProteinHit proteinHit = new ProteinHit();
105      proteinHits.add(proteinHit);
106
```

```
107         proteinHit.expect = msProtein.getProtExpect();
108         proteinHit.score = msProtein.getProtScore();
109         proteinHit.acc = msProtein.getAccession();
110         proteinHit.desc = msProtein.getProtDesc();
111         proteinHit.matches = msProtein.getProtMatches();
112         proteinHit.coverage = msProtein.getProtCover();
113         for (PeptideType msPeptide : msProtein.getPeptideArray())
114         {
115          proteinHit.PepFragments += msPeptide.getPepNumMatch();
116          // String msPepVarMod = msPeptide.getPepVarMod();
117
118          String[] scanTitleArr = { "0", "0", "0", "0", "0", "0" };
119          if (msPeptide.getPepScanTitle().matches(
120            "\\d+" + SCANTITLE_SEP_CHAR + "\\d*\\.?\\d+"
121              + SCANTITLE_SEP_CHAR + "\\*+\\.?\\d+"
122              + SCANTITLE_SEP_CHAR + "\\w+"
123              + SCANTITLE_SEP_CHAR + "\\d*\\.\\d+"
124              + SCANTITLE_SEP_CHAR + "\\d*\\.?\\d"))
125          {
126           scanTitleArr = msPeptide.getPepScanTitle().split(
127             SCANTITLE_SEP_CHAR);
128          }
129          int scan = Integer
130            .parseInt(scanTitleArr[SCANTITLE_SCAN]);
131          double calcNeutralMass = msPeptide.getPepCalcMr();
132          BigInteger charge = msPeptide.getPepExpZ();
133          String sequence = msPeptide.getPepSeq();
134          String proteinName = msProtein.getProtDesc();
135          List<ModifiedAminoAcid>[] modificationListArray = null;
136          Feature currentFeature = MS2ExtraInfoDef
137            .createMS2Feature(scan,
138              (float) calcNeutralMass,
139              charge.intValue(), sequence,
140              proteinName, modificationListArray);
141
142          scoreMap.put(IONSCORE, Double.toString(msPeptide
143            .getPepScore()));
144          scoreMap.put(IDENTITYSCORE, Double.toString(msPeptide
145            .getPepIdent()));
146          scoreMap.put(EXPECT, Double.toString(msPeptide
147            .getPepExpect()));
148          MS2ExtraInfoDef.setDeltaMass(currentFeature,
149            (float) msPeptide.getPepDelta());
150          MS2ExtraInfoDef.setSearchScores(currentFeature,
151            scoreMap);
152
153          String prevAA = msPeptide.getPepResBefore();
154          String nextAA = msPeptide.getPepResAfter();
155          if (prevAA != null)
156           MS2ExtraInfoDef.setPrevAminoAcid(currentFeature,
157             prevAA.charAt(0));
158          if (nextAA != null)
159           MS2ExtraInfoDef.setNextAminoAcid(currentFeature,
160             nextAA.charAt(0));
161
162          if (msr.getSearchParameters().getCLE()
163            .equalsIgnoreCase("Trypsin"))
164          {
165           if (prevAA != null
166             && (prevAA.startsWith("K") || prevAA
167               .startsWith("R")))
168           {
169            int numTrypticEnds = 1;
170            if (sequence.endsWith("K")
171              || sequence.endsWith("R"))
172             numTrypticEnds++;
173            MS2ExtraInfoDef.setNumEnzymaticEnds(
174              currentFeature, numTrypticEnds);
175           }
176          }
177
178          MS2ExtraInfoDef.setNumFragments(currentFeature,
179            (int) msPeptide.getPepNumMatch());
180
```

```
181         currentFeature.setTime(Float
182           .parseFloat(scanTitleArr[SCANTITLE_TIME]));
183
184         proteinHit.add(currentFeature);
185       }
186      }
187     }
188    } catch (XmlException e)
189    {
190     throw new IOException("Unable to load '" + file + "'", e);
191    }
192    return proteinHits;
193   }
194
195   /**
196    * Load a FeatureSet
197    *
198    * @param file
199    * @return
200    * @throws IOException
201    */
202   public FeatureSet loadFeatureSet(File file) throws IOException
203   {
204    List<Feature> listOfAllPeptides = new ArrayList<Feature>();
205    List<ProteinHit> proteinHits = loadProteinHits(file);
206    for (ProteinHit proteinHit : proteinHits)
207    {
208     listOfAllPeptides.addAll(proteinHit.getPeptideMatches());
209    }
210    return new FeatureSet(listOfAllPeptides.toArray(new Feature[] {}));
211   }
212
213   protected void setFeatureSetPropertiesFromFraction(MascotSearchResults
         msr,
214     FeatureSet featureSet)
215   {
216    // MS2Modification[] modifications =
217    // fraction.getModifications().toArray(
218    // new MS2Modification[0]);
219    // if (modifications != null && modifications.length > 0)
220    // {
221    // for (MS2Modification ms2Mod : modifications)
222    // MS2ExtraInfoDef.updateMS2ModMassOrDiff(ms2Mod);
223    // }
224    // MS2ExtraInfoDef.correctMS2ModMasses(modifications);
225    // _log.debug("\tloaded "
226    // + (modifications == null ? 0 : modifications.length
227    // + " MS2 modifications"));
228    // MS2ExtraInfoDef.setFeatureSetModifications(featureSet,
229    // modifications);
230
231    SearchParameters msParameters = msr.getSearchParameters();
232    int maxCleavages = msParameters.getPFA();
233    if (maxCleavages > 0)
234    {
235     MS2ExtraInfoDef.setFeatureSetSearchConstraintMaxIntCleavages(
236       featureSet, maxCleavages);
237    }
238    int minTermini = 0;
239    if (msParameters.getCLE().equalsIgnoreCase("Trypsin"))
240    {
241     minTermini = 2;
242    }
243    if (minTermini > 0)
244    {
245     MS2ExtraInfoDef.setFeatureSetSearchConstraintMinTermini(featureSet,
246       minTermini);
247    }
248
249    Header msHeader = msr.getHeader();
250    String databaseName = msHeader.getDB();
251    if (databaseName != null)
252    {
253     MS2ExtraInfoDef.setFeatureSetBaseName(featureSet, databaseName);
```

```
254     }
255     _log.debug("\tmin termini=" + minTermini + ", max cleavages="
256       + maxCleavages);
257
258   }
259
260   public void saveFeatureSet(FeatureSet featureSet, File outFile)
261     throws IOException
262   {
263     FeaturePepXmlWriter pepXmlWriter = new FeaturePepXmlWriter(featureSet)
          ;
264     pepXmlWriter.setFirstSpectrumQueryIndex(firstSpectrumQueryIndex);
265     try
266     {
267       pepXmlWriter.write(outFile);
268     } catch (Exception e)
269     {
270       _log.error("Failed to save pepXML", e);
271     }
272   }
273
274   /**
275    * Save a FeatureSet
276    *
277    * @param featureSet
278    * @param out
279    */
280   public void saveFeatureSet(FeatureSet featureSet, PrintWriter out)
281   {
282     throw new IllegalArgumentException(
283       "This version of saveFeatureSet not implemented in
              MascotXMLFeatureFileHandler");
284   }
285
286   /**
287    * Can this type of file handler handle this specific file?
288    *
289    * @param file
290    * @return
291    * @throws IOException
292    */
293   public boolean canHandleFile(File file) throws IOException
294   {
295
296     if (!isXMLFile(file))
297     {
298       _log.debug("canHandleFile, File is not XML");
299       return false;
300     }
301     _log.debug("canHandleFile, File is XML...");
302     FileInputStream fis = null;
303     boolean result = false;
304     try
305     {
306       fis = new FileInputStream(file);
307       SimpleXMLStreamReader parser = new SimpleXMLStreamReader(fis);
308
309       while (!parser.isStartElement())
310         parser.next();
311       String startElementName = parser.getLocalName();
312
313       // check that the first element is an msms_pipeline_analysis. I'm
314       // pretty
315       // sure that this is required by the pepXML spec, but whether it is
316       // or not,
317       // if we run into files where this doesn't hold true, will need to
318       // change.
319       if ("mascot_search_results".equalsIgnoreCase(startElementName))
320         result = true;
321       else
322       {
323         _log
324           .debug("canHandleFile, First element is not mascot_search_results
                ... it's "
```

```
325             + startElementName);
326       return false;
327     }
328   } catch (XMLStreamException xse)
329   {
330     _log.debug("canHandleFile, throwing exception with message "
331       + xse.getMessage());
332     throw new IOException(xse.getMessage());
333   } finally
334   {
335     if (fis != null)
336       fis.close();
337   }
338   _log.debug("canHandleFile, returning true!");
339   return result;
340 }
341
342 public int getFirstSpectrumQueryIndex()
343 {
344  return firstSpectrumQueryIndex;
345 }
346
347 public void setFirstSpectrumQueryIndex(int firstSpectrumQueryIndex)
348 {
349  this.firstSpectrumQueryIndex = firstSpectrumQueryIndex;
350 }
351
352 public class ProteinHit
353 {
354  private long PepFragments;
355  private double coverage;
356  private long matches;
357  private String desc;
358  private String acc;
359  private List<Feature> peptideMatches;
360  private double rms = -1d;
361  private double score = -1d;
362  private double expect = -1d;
363
364  private ProteinHit()
365  {
366   peptideMatches = new ArrayList<Feature>();
367  }
368
369  private void add(Feature feature)
370  {
371   peptideMatches.add(feature);
372  }
373
374  public long getPepFragments()
375  {
376   return PepFragments;
377  }
378
379  public double getCoverage()
380  {
381   return coverage;
382  }
383
384  public long getMatches()
385  {
386   return matches;
387  }
388
389  public String getDesc()
390  {
391   return desc;
392  }
393
394  public String getAcc()
395  {
396   return acc;
397  }
398
```

```
399     public List<Feature> getPeptideMatches()
400     {
401      return peptideMatches;
402     }
403
404     public double getRms()
405     {
406      if (rms < 0)
407      {
408       rms = 0;
409       for (Feature pep : getPeptideMatches())
410       {
411        rms += Math.pow(MS2ExtraInfoDef.getDeltaMass(pep) / pep.mass *
               1000000, 2);
412       }
413       rms = Math.sqrt(rms/getPeptideMatches().size());
414      }
415      return rms;
416     }
417
418     public double getScore()
419     {
420      return score;
421     }
422
423     public double getExpect()
424     {
425      return expect;
426     }
427    }
428   }
```

Listing E.9: Plots the methods and proteins versus response

```
1   ####################
2   # Method plotting
3   plotmethods <- function(data, response, minmax, medians, legend = TRUE)
        {
4   methodMatches <- data[,c("method.nr",response)]
5   mm <- methodMatches
6   if (medians) {
7   plotme <- aggregate(mm[response], list(Method.nr = mm$method.nr), median
        )
8   lineme <- aggregate(mm[response], list(Method.nr = mm$method.nr), mean)
9   } else {
10  plotme <- aggregate(mm[response], list(Method.nr = mm$method.nr), mean)
11  lineme <- aggregate(mm[response], list(Method.nr = mm$method.nr), median
        )
12  }
13  plot(plotme, type="p", ylim=c(0,max(mm[response])))
14  lines(lineme, type="p", pch="-", cex=4,  col="grey")
15  if (minmax) {
16  maxima <- aggregate(mm[response], list(Method.nr = mm$method.nr), max)
17  lines(maxima, type="l", lty="dotted")
18  minima <- aggregate(mm[response], list(Method.nr = mm$method.nr), min)
19  lines(minima, type="l", lty="dotted")
20  }
21  title(response)
22
23  i <- 0
24  for (protnr in levels(data$Prot.nr)) {
25   i <- i + 1
26   methodMatches <- data[data$Prot.nr==protnr,c("method.nr",response)]
27   mm <- methodMatches
28   if (medians) {
29    means <- aggregate(mm[response], list(Method.nr = mm$method.nr),
          median)
30   } else {
31    means <- aggregate(mm[response], list(Method.nr = mm$method.nr), mean)
32   }
33   lines(means, type="o", col=proteincolors[i])
34   if (minmax) {
35    maxima <- aggregate(mm[response], list(Method.nr = mm$method.nr), max)
```

```
36    lines(maxima, type="l", lty="dotted", col=proteincolors[i])
37    minima <- aggregate(mm[response], list(Method.nr = mm$method.nr), min)
38    lines(minima, type="l", lty="dotted", col=proteincolors[i])
39   }
40  }
41  if (legend) {
42   legend("topright", levels(data$Prot.nr), fill = proteincolors)
43  }
44  }
45
46  ###################
47  # Protein plots
48  #response <- "Matches"
49  plotproteins <- function(data, response, minmax, medians, legend = TRUE)
        {
50  methodMatches <- data[,c("Prot.nr",response)]
51  mm <- methodMatches
52  if (medians) {
53  plotme <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), median)
54  lineme <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), mean)
55  } else {
56  plotme <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), mean)
57  lineme <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), median)
58  }
59  plot(plotme, type="p", ylim=c(0,max(mm[response])), legend = TRUE)
60  lines(lineme, type="p", pch="-", cex=4,  col="grey")
61  if (minmax) {
62  maxima <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), max)
63  lines(maxima, type="l", lty="dotted")
64  minima <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), min)
65  lines(minima, type="l", lty="dotted")
66  }
67  title(response)
68
69  i = 0
70  for (methodnr in levels(data$method.nr)) {
71   i <- i + 1
72   methodMatches <- data[data$method.nr==methodnr,c("Prot.nr",response)]
73   mm <- methodMatches
74   if (medians) {
75    means <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), median
          )
76   } else {
77    means <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), mean)
78   }
79   lines(means, type="o", col=methodcolors[i])
80   if (minmax) {
81    maxima <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), max)
82    lines(maxima, type="l", lty="dotted", col=methodcolors[i])
83    minima <- aggregate(mm[response], list(Protein.nr = mm$Prot.nr), min)
84    lines(minima, type="l", lty="dotted", col=methodcolors[i])
85   }
86  }
87  if (legend) {
88   legend("topright", levels(data$method.nr), fill = methodcolors)
89  }
90  }
91
92
93  ###########################################################
94  # Use the methods
95
96
97  data <- read.delim("e:\\mascot_stuff\\allealle_samlet.protein.tsv")
98  #fix(data)
99
100 tmp <- data
101 #data <- data[data$method.nr %in% c(1:4,6,9,10,5),]
102
103 #lm <- lm(data$Matches ~ as.factor(data$method) * as.factor(data$
        Accession))
104 #summary(lm)
105 #anova(lm)
106
```

```
minmax = FALSE
medians = FALSE

responses = c("Matches", "Coverage", "Peptide.RMS", "Fragments", "Score")

#####################
# Anovas
for (response in responses) {
 formula <- as.formula(paste(response, " ~ method * Accession"))
 print(anova(lm(formula, data = data)))
# print(anova(lm(formula, data = data[data$method.nr %in% c("n1", "n2", "n3", "n4", "n5", "n6"),])))
}

par(mfrow=c(2,3))

#####################
# Method plots

proteincolors <- c("green", "red", "blue", "olivedrab")
for (response in responses) {
 plotmethods(data, response, minmax, medians)
}
plot.new()
legend("topright", paste(levels(data$Prot.nr), " = ", levels(data$Accession)), fill = proteincolors)

par(mfrow=c(2,3))

###################
# Protein plots
methodcolors = c("pink", "green", "red", "blue", "violet", "turquoise", "peru", "olivedrab", "skyblue4")
for (response in responses) {
 plotproteins(data, response, minmax, medians)
}
plot.new()
legend("topright", paste(levels(data$method.nr), "=", levels(data$method)), fill = methodcolors)


par(mfrow=c(1,2))
plotmethods(data, "Matches", minmax, medians, legend = FALSE)
legend("topright", levels(data$Accession), fill = proteincolors)
plot.new()
legend("topright", paste(levels(data$method.nr), "=", levels(data$method)))
```

Listing E.10: Calculates the protein ratios, given a file of peptide intensities (from MyPepArrayAnalyzer)

```
1  data <- read.delim("e:\\raw\\Iain mpds\\250208_mix1-2_004-009.mzXML.peparray.out.tsv", comment.char = "#");
2  #summary(data)
3
4  ##############
5  # Functions
6
7  div <- function(x) {
8  x[1] / x[2]
9  }
10
11 calcweirdratio <- function(data) {
12  aprotmean <- aggregate(data$intensity, list(protein = data$protein, caseorcontrol = data$caseorcontrol), mean)
13  aprotmedian <- aggregate(data$intensity, list(protein = data$protein, caseorcontrol = data$caseorcontrol), median)
14  aprotmeanratio <- aggregate(aprotmean$x, list(protein = aprotmean$protein), div)
15  aprotmedianratio <- aggregate(aprotmedian$x, list(protein = aprotmedian$protein), div)
16 # aprotsd <- aggregate(data$intensity, list(protein = data$protein), sd)
```

```
17 # aprotmean2 <- aggregate(data$intensity, list(protein = data$protein),
       mean)
18 #, cv = aprotsd$x/aprotmean2$x*100
19   signif(data.frame(protein = aprotmeanratio$protein, meanratio =
       aprotmeanratio$x, medianratio = aprotmedianratio$x)[,-1],3)
20 }
21
22 calcprotratio <- function(data) {
23  apepmean <- aggregate(data$intensity, list(peptide = data$peptide,
       protein = data$protein, caseorcontrol = data$caseorcontrol), mean)
24  apepmedian <- aggregate(data$intensity, list(peptide = data$peptide,
       protein = data$protein, caseorcontrol = data$caseorcontrol),
       median)
25  apepmeanratio <- aggregate(apepmean$x, list(peptide = apepmean$peptide,
        protein = apepmean$protein), div)
26  apepmedianratio <- aggregate(apepmedian$x, list(peptide = apepmedian$
       peptide, protein = apepmedian$protein), div)
27  aprotratiomean <- aggregate(apepmeanratio$x, list(protein =
       apepmeanratio$protein), mean)
28  aprotratiomeansd <- aggregate(apepmeanratio$x, list(protein =
       apepmeanratio$protein), sd)
29  aprotratiomeaniqr <- aggregate(apepmeanratio$x, list(protein =
       apepmeanratio$protein), IQR)
30  aprotratiomeanmad <- aggregate(apepmeanratio$x, list(protein =
       apepmeanratio$protein), mad)
31  aprotratiomedianmean <- aggregate(apepmeanratio$x, list(protein =
       apepmeanratio$protein), median)
32  aprotratiomedian <- aggregate(apepmedianratio$x, list(protein =
       apepmedianratio$protein), median)
33  aprotratiomediansd <- aggregate(apepmedianratio$x, list(protein =
       apepmedianratio$protein), sd)
34  aprotratiomedianiqr <- aggregate(apepmedianratio$x, list(protein =
       apepmedianratio$protein), IQR)
35  aprotratiomedianmad <- aggregate(apepmedianratio$x, list(protein =
       apepmedianratio$protein), mad)
36  aprotratiomeanmedian <- aggregate(apepmedianratio$x, list(protein =
       apepmedianratio$protein), mean)
37 # aprotratiologmean <- aggregate(log(apepmeanratio$x), list(protein =
       apepmeanratio$protein), mean)
38 # aprotratiologsd <- aggregate(log(apepmeanratio$x), list(protein =
       apepmeanratio$protein), sd)
39 # aprotratiologsd$x <- log(aprotratiosd$x)
40 #, logratio = aprotratiologmean$x, logsd = aprotratiologsd$x
41   signif(data.frame(protein = aprotratiomean$protein, meanratio =
       aprotratiomean$x, medianmeanratio = aprotratiomedianmean$x, meansd
        = aprotratiomeansd$x, meaniqr = aprotratiomeaniqr$x,
       aprotratiomedian = aprotratiomedian$x, aprotratiomeanmedian =
       aprotratiomeanmedian$x, mediansd = aprotratiomediansd$x, medianiqr
        = aprotratiomedianiqr$x)[,-1],3)
42 }
43
44 percentdiff <- function(x) {
45 (x[1] - x[2]) / x[2] * 100
46 }
47
48 ##########################
49 # All peptides
50 calcweirdratio(data)
51 calcprotratio(data)
52
53 ########################################################
54 # Choosing only those which are seen in all runs
55 runcount <- aggregate(data$intensity, list(peptide = data$peptide,
       protein = data$protein), length)
56 allruns <- data[(data$peptide %in% runcount[runcount$x==6, "peptide"]) |
       (data$peptide %in% runcount[runcount$x==12, "peptide"]),]
57 #summary(allruns)
58 calcweirdratio(allruns)
59 calcprotratio(allruns)
60 ##########################
61 # All peptides, totalIntensities
62 totalIntensities <- data
63 totalIntensities$intensity <- data$totalintensity
64 calcweirdratio(totalIntensities)
```

```
65  calcprotratio(totalIntensities)
66
67  ######################################################
68  # Choosing only those which are seen in all runs, totalIntensities
69  totalIntensitiesAll <- allruns
70  totalIntensitiesAll$intensity <- allruns$totalintensity
71  calcweirdratio(totalIntensitiesAll)
72  calcprotratio(totalIntensitiesAll)
73
74  ###################
75  #rms of methods
76  ratios <- c(1,0.5,2,8)
77  rms <- function(data) {
78   signif(sqrt(colSums((data[1:4,] - ratios[1:4])^2)/length(ratios[1:4]))
         ,3)
79  }
80  cols <- c(-3,-6)
81  a<-rbind(
82  rms(calcprotratio(data)[,cols]),
83  rms(calcprotratio(allruns)[,cols]),
84  rms(calcprotratio(totalIntensities)[,cols]),
85  rms(calcprotratio(totalIntensitiesAll)[,cols])
86  )
87  b<-rbind(
88  rms(calcweirdratio(data)),
89  rms(calcweirdratio(allruns)),
90  rms(calcweirdratio(totalIntensities)),
91  rms(calcweirdratio(totalIntensitiesAll))
92  )
93
94  ############################
95  # rms without ALBU_BOVIN
96  rms <- function(data) {
97   signif(sqrt(colSums((data[1:3,] - ratios[1:3])^2)/length(ratios[1:3]))
         ,3)
98  }
99  cols <- c(-3,-6)
100 c<-rbind(
101 rms(calcprotratio(data)[,cols]),
102 rms(calcprotratio(allruns)[,cols]),
103 rms(calcprotratio(totalIntensities)[,cols]),
104 rms(calcprotratio(totalIntensitiesAll)[,cols])
105 )
106 d<-rbind(
107 rms(calcweirdratio(data)),
108 rms(calcweirdratio(allruns)),
109 rms(calcweirdratio(totalIntensities)),
110 rms(calcweirdratio(totalIntensitiesAll))
111 )
112
113 ###################
114 #relative rms of methods
115 ratios <- c(1,0.5,2,8)
116 relativerms <- function(data) {
117  signif(sqrt(colSums((data[1:4,] / ratios[1:4] - 1)^2)/length(ratios
       [1:4])),3)
118 }
119 cols <- c(-3,-6)
120 e<-rbind(
121 relativerms(calcprotratio(data)[,cols]),
122 relativerms(calcprotratio(allruns)[,cols]),
123 relativerms(calcprotratio(totalIntensities)[,cols]),
124 relativerms(calcprotratio(totalIntensitiesAll)[,cols])
125 )
126 f<-rbind(
127 relativerms(calcweirdratio(data)),
128 relativerms(calcweirdratio(allruns)),
129 relativerms(calcweirdratio(totalIntensities)),
130 relativerms(calcweirdratio(totalIntensitiesAll))
131 )
132
133 ############################
134 # rms without ALBU_BOVIN
135 relativerms <- function(data) {
```

```
136   signif(sqrt(colSums((data[1:3,] / ratios[1:3] - 1)^2)/length(ratios
          [1:3])),3)
137 }
138 cols <- c(-3,-6)
139 g<-rbind(
140 relativerms(calcprotratio(data)[,cols]),
141 relativerms(calcprotratio(allruns)[,cols]),
142 relativerms(calcprotratio(totalIntensities)[,cols]),
143 relativerms(calcprotratio(totalIntensitiesAll)[,cols])
144 )
145 h<-rbind(
146 relativerms(calcweirdratio(data)),
147 relativerms(calcweirdratio(allruns)),
148 relativerms(calcweirdratio(totalIntensities)),
149 relativerms(calcweirdratio(totalIntensitiesAll))
150 )
151
152
153 a
154 b
155 c
156 d
157 ##### relative
158 e
159 f
160 g
161 h
162
163
164 ##################
165 # PGLS RMS
166 auto <- c(11.47,3.22,.82,1.58)[4:1]
167 nonorm <- c(7.92,2.1,.54,1.01)[4:1]
168 stdnorm <- c(7.17,2.01,.52)[3:1]
169 #######
170 # Std dev
171 stddev <- c(.04,.03,.07,.04)
172 (exp(log(nonorm)+stddev)-exp(log(nonorm)-stddev))/2
173
174 signif(c(
175 sqrt(sum((auto - ratios)^2)/length(ratios)),
176 sqrt(sum((nonorm - ratios)^2)/length(ratios)),
177 sqrt(sum((stdnorm - ratios[2:4])^2)/length(ratios[1:3]))
178 ),3)
179 signif(c(
180 sqrt(sum((auto[1:3] - ratios[1:3])^2)/length(ratios[1:3])),
181 sqrt(sum((nonorm[1:3] - ratios[1:3])^2)/length(ratios[1:3])),
182 sqrt(sum((stdnorm[1:2] - ratios[2:3])^2)/length(ratios[1:2]))
183 ),3)
184
185 ##################
186 # PGLS relative RMS
187 auto <- c(11.47,3.22,.82,1.58)[4:1]
188 nonorm <- c(7.92,2.1,.54,1.01)[4:1]
189 stdnorm <- c(7.17,2.01,.52)[3:1]
190
191 signif(c(
192 sqrt(sum((auto / ratios - 1)^2)/length(ratios)),
193 sqrt(sum((nonorm / ratios - 1)^2)/length(ratios)),
194 sqrt(sum((stdnorm / ratios[2:4] - 1)^2)/length(ratios[1:3]))
195 ),3)
196 signif(c(
197 sqrt(sum((auto[1:3] / ratios[1:3] - 1)^2)/length(ratios[1:3])),
198 sqrt(sum((nonorm[1:3] / ratios[1:3] - 1)^2)/length(ratios[1:3])),
199 sqrt(sum((stdnorm[1:2] / ratios[2:3] - 1)^2)/length(ratios[1:2]))
200 ),3)
201
202 ##################
203 # PLGS +E. Coli RMS
204 auto <- c(6.11,1.84,.48,.89)[4:1]
205 nonorm <- c(7.1,2.12,.54,1.08)[4:1]
206 stdnorm <- c(6.75,2.05,.53)[3:1]
207 #######
208 # Std dev
```

```
stddev <- c(.03,.03,.04,.03)
(exp(log(nonorm)+stddev)-exp(log(nonorm)-stddev))/2

signif(c(
sqrt(sum((auto - ratios)^2)/length(ratios)),
sqrt(sum((nonorm - ratios)^2)/length(ratios)),
sqrt(sum((stdnorm - ratios[2:4])^2)/length(ratios[1:3]))
),3)
signif(c(
sqrt(sum((auto[1:3] - ratios[1:3])^2)/length(ratios[1:3])),
sqrt(sum((nonorm[1:3] - ratios[1:3])^2)/length(ratios[1:3])),
sqrt(sum((stdnorm[1:2] - ratios[2:3])^2)/length(ratios[1:2]))
),3)

##################
# PLGS +E. Coli relative RMS
auto <- c(6.11,1.84,.48,.89)[4:1]
nonorm <- c(7.1,2.12,.54,1.08)[4:1]
stdnorm <- c(6.75,2.05,.53)[3:1]

signif(c(
sqrt(sum((auto / ratios - 1)^2)/length(ratios)),
sqrt(sum((nonorm / ratios - 1)^2)/length(ratios)),
sqrt(sum((stdnorm / ratios[2:4] - 1)^2)/length(ratios[1:3]))
),3)
signif(c(
sqrt(sum((auto[1:3] / ratios[1:3] - 1)^2)/length(ratios[1:3])),
sqrt(sum((nonorm[1:3] / ratios[1:3] - 1)^2)/length(ratios[1:3])),
sqrt(sum((stdnorm[1:2] / ratios[2:3] - 1)^2)/length(ratios[1:2]))
),3)


########################
# Test averages

apepmean <- aggregate(data$intensity, list(peptide = data$peptide,
    protein = data$protein, caseorcontrol = data$caseorcontrol), mean)
apepmeanratio <- aggregate(apepmean$x, list(peptide = apepmean$peptide,
    protein = apepmean$protein), div)

average <- function(data) {
#geometric
 prod(data)^(1/length(data))
}
aggregate(apepmeanratio$x, list(protein = apepmeanratio$protein),
    average)
average <- function(data) {
#harmonic
 length(data)/sum(1/data)
}
aggregate(apepmeanratio$x, list(protein = apepmeanratio$protein),
    average)
average <- function(data) {
#generalized
 p <- 2
 (sum(data^p)/length(data))^(1/p)
}
aggregate(apepmeanratio$x, list(protein = apepmeanratio$protein),
    average)
average <- function(data) {
#generalized
 p <- 3
 (sum(data^p)/length(data))^(1/p)
}
aggregate(apepmeanratio$x, list(protein = apepmeanratio$protein),
    average)
average <- function(data) {
#truncated
 p <- 0.6
 n <- length(data)
# mean(data[(n*(1-p)/2):(n*(1-(1-p)/2))])
 mean(data, trim=(1-p)/2)
}
aggregate(apepmeanratio$x, list(protein = apepmeanratio$protein),
```

```
    average)
277 average <- function(data) {
278 #winsorized
279  p <- 0.6
280  n <- length(data)
281  start <- (n*(1-p)/2)
282  end <- (n*(1-(1-p)/2))
283  tmp <- data[start:end]
284  mean(c(rep(tmp[1], start-1), tmp, rep(tmp[length(tmp)], length(data)-
         end)))
285 }
286 aggregate(apepmeanratio$x, list(protein = apepmeanratio$protein),
        average)
```

Listing E.11: msInspect script for generating calibrated peak lists from mzXML files (Iain dataset)

```
 1 findpeptides
 2  count=2147483647
 3  featurestrategy=FeatureStrategyPeakClusters
 4  accuratemassscans=3
 5  outdir=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
 6  noaccuratemass=false
 7  dumpwindow=0
 8  start=1
 9  maxmz=1300
10  C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds\250208
        _mix1_004.mzXML,C:\Documents and Settings\rschj02\My Documents\Raw
        \Iain mpds\250208_mix1_005.mzXML,C:\Documents and Settings\rschj02
        \My Documents\Raw\Iain mpds\250208_mix1_006.mzXML
11  plotstats=false
12  walksmoothed=false
13 findpeptides
14  count=2147483647
15  featurestrategy=FeatureStrategyMSe_lvl2
16  accuratemassscans=3
17  outdir=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
18  noaccuratemass=false
19  dumpwindow=0
20  start=1
21  maxmz=1300
22  C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds\250208
        _mix1_004.mzXML,C:\Documents and Settings\rschj02\My Documents\Raw
        \Iain mpds\250208_mix1_005.mzXML,C:\Documents and Settings\rschj02
        \My Documents\Raw\Iain mpds\250208_mix1_006.mzXML
23  suffix=.peptides_ms2.tsv
24  plotstats=false
25  walksmoothed=false
26 CalibrationUsingLockspray
27  featureout=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_004.peptides.recalib.tsv
28  featuresetfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain
        mpds\250208_mix1_004.peptides.tsv
29  plotcalibration=false
30  mzxmlfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_004.mzXML
31 CalibrationUsingLockspray
32  featureout=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_005.peptides.recalib.tsv
33  featuresetfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain
        mpds\250208_mix1_005.peptides.tsv
34  plotcalibration=false
35  mzxmlfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_005.mzXML
36 CalibrationUsingLockspray
37  featureout=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_006.peptides.recalib.tsv
38  featuresetfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain
        mpds\250208_mix1_006.peptides.tsv
39  plotcalibration=false
40  mzxmlfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_006.mzXML
41 CalibrationUsingLockspray
```

```
42  featureout=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_004.peptides_ms2.recalib.tsv
43  featuresetfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain
        mpds\250208_mix1_004.peptides_ms2.tsv
44  plotcalibration=false
45  mzxmlfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_004.mzXML
46 CalibrationUsingLockspray
47  featureout=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_005.peptides_ms2.recalib.tsv
48  featuresetfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain
        mpds\250208_mix1_005.peptides_ms2.tsv
49  plotcalibration=false
50  mzxmlfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_005.mzXML
51 CalibrationUsingLockspray
52  featureout=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_006.peptides_ms2.recalib.tsv
53  featuresetfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain
        mpds\250208_mix1_006.peptides_ms2.tsv
54  plotcalibration=false
55  mzxmlfile=C:\Documents and Settings\rschj02\My Documents\Raw\Iain mpds
        \250208_mix1_006.mzXML
56 MSeFeatureSetCombiner
57  precursors=E:\Raw\Iain mpds\250208_mix1_004.peptides.recalib.tsv
58  fragments=E:\Raw\Iain mpds\250208_mix1_004.peptides_ms2.recalib.tsv
59  outfile=E:\Raw\Iain mpds\250208_mix1_004.mzXML.pkl
60  mzxmlfile=E:\Raw\Iain mpds\250208_mix1_004.mzXML
61  formatmaker=PKL
62 MSeFeatureSetCombiner
63  precursors=E:\Raw\Iain mpds\250208_mix1_004.peptides.recalib.tsv
64  fragments=E:\Raw\Iain mpds\250208_mix1_004.peptides_ms2.recalib.tsv
65  outfile=E:\Raw\Iain mpds\250208_mix1_004.mzXML.mgf
66  mzxmlfile=E:\Raw\Iain mpds\250208_mix1_004.mzXML
67  formatmaker=MGF
68 MSeFeatureSetCombiner
69  precursors=E:\Raw\Iain mpds\250208_mix1_005.peptides.recalib.tsv
70  fragments=E:\Raw\Iain mpds\250208_mix1_005.peptides_ms2.recalib.tsv
71  outfile=E:\Raw\Iain mpds\250208_mix1_005.mzXML.pkl
72  mzxmlfile=E:\Raw\Iain mpds\250208_mix1_005.mzXML
73  formatmaker=PKL
74 MSeFeatureSetCombiner
75  precursors=E:\Raw\Iain mpds\250208_mix1_005.peptides.recalib.tsv
76  fragments=E:\Raw\Iain mpds\250208_mix1_005.peptides_ms2.recalib.tsv
77  outfile=E:\Raw\Iain mpds\250208_mix1_005.mzXML.mgf
78  mzxmlfile=E:\Raw\Iain mpds\250208_mix1_005.mzXML
79  formatmaker=MGF
80 MSeFeatureSetCombiner
81  precursors=E:\Raw\Iain mpds\250208_mix1_006.peptides.recalib.tsv
82  fragments=E:\Raw\Iain mpds\250208_mix1_006.peptides_ms2.recalib.tsv
83  outfile=E:\Raw\Iain mpds\250208_mix1_006.mzXML.pkl
84  mzxmlfile=E:\Raw\Iain mpds\250208_mix1_006.mzXML
85  formatmaker=PKL
86 MSeFeatureSetCombiner
87  precursors=E:\Raw\Iain mpds\250208_mix1_006.peptides.recalib.tsv
88  fragments=E:\Raw\Iain mpds\250208_mix1_006.peptides_ms2.recalib.tsv
89  outfile=E:\Raw\Iain mpds\250208_mix1_006.mzXML.mgf
90  mzxmlfile=E:\Raw\Iain mpds\250208_mix1_006.mzXML
91  formatmaker=MGF
92 MSeFeatureSetCombiner
93  precursors=E:\Raw\Iain mpds\250208_mix2_007.peptides.recalib.tsv
94  fragments=E:\Raw\Iain mpds\250208_mix2_007.peptides_ms2.recalib.tsv
95  outfile=E:\Raw\Iain mpds\250208_mix2_007.mzXML.pkl
96  mzxmlfile=E:\Raw\Iain mpds\250208_mix2_007.mzXML
97  formatmaker=PKL
98 MSeFeatureSetCombiner
99  precursors=E:\Raw\Iain mpds\250208_mix2_007.peptides.recalib.tsv
100 fragments=E:\Raw\Iain mpds\250208_mix2_007.peptides_ms2.recalib.tsv
101 outfile=E:\Raw\Iain mpds\250208_mix2_007.mzXML.mgf
102 mzxmlfile=E:\Raw\Iain mpds\250208_mix2_007.mzXML
103 formatmaker=MGF
104 MSeFeatureSetCombiner
105 precursors=E:\Raw\Iain mpds\250208_mix2_008.peptides.recalib.tsv
106 fragments=E:\Raw\Iain mpds\250208_mix2_008.peptides_ms2.recalib.tsv
```

```
107   outfile=E:\Raw\Iain mpds\250208_mix2_008.mzXML.pkl
108   mzxmlfile=E:\Raw\Iain mpds\250208_mix2_008.mzXML
109   formatmaker=PKL
110 MSeFeatureSetCombiner
111   precursors=E:\Raw\Iain mpds\250208_mix2_008.peptides.recalib.tsv
112   fragments=E:\Raw\Iain mpds\250208_mix2_008.peptides_ms2.recalib.tsv
113   outfile=E:\Raw\Iain mpds\250208_mix2_008.mzXML.mgf
114   mzxmlfile=E:\Raw\Iain mpds\250208_mix2_008.mzXML
115   formatmaker=MGF
116 MSeFeatureSetCombiner
117   precursors=E:\Raw\Iain mpds\250208_mix2_009.peptides.recalib.tsv
118   fragments=E:\Raw\Iain mpds\250208_mix2_009.peptides_ms2.recalib.tsv
119   outfile=E:\Raw\Iain mpds\250208_mix2_009.mzXML.pkl
120   mzxmlfile=E:\Raw\Iain mpds\250208_mix2_009.mzXML
121   formatmaker=PKL
122 MSeFeatureSetCombiner
123   precursors=E:\Raw\Iain mpds\250208_mix2_009.peptides.recalib.tsv
124   fragments=E:\Raw\Iain mpds\250208_mix2_009.peptides_ms2.recalib.tsv
125   outfile=E:\Raw\Iain mpds\250208_mix2_009.mzXML.mgf
126   mzxmlfile=E:\Raw\Iain mpds\250208_mix2_009.mzXML
127   formatmaker=MGF
```