1 Introduction

The \textit{yamss} (yet another mass spectrometry software) package provides tools to preprocess raw mass spectral data files arising from metabolomics experiments with the primary goal of providing high-quality differential analysis. Currently, \textit{yamss} implements a preprocessing method “\textit{bakedpi}”, which stands for bivariate approximate kernel density estimation for peak identification.

Alternatives to this package include \textit{xcms} (Smith et al. 2006) (available on Bioconductor) and MZMine 2 (Pluskal et al. 2010). These packages also provide preprocessing functionality but focus more on feature detection and alignment in individual samples. The input data to \textit{yamss} are standard metabolomics mass spectral files which can be read by \textit{mzR}.

1.1 \textit{bakedpi}

The “\textit{bakedpi}” algorithm is a new preprocessing algorithm for untargeted metabolomics data (manuscript in preparation). The output of “\textit{bakedpi}” is essentially a table with dimension peaks (adducts) by samples, containing quantified intensity measurements representing the abundance of metabolites. This table, which is very similar to output in gene expression analysis, can directly be used in standard statistical packages, such as \textit{limma}, to identify differentially abundant metabolites between conditions. It is critical that all samples which are analyzed together, are processed together through \textit{bakedpi}. 

1.2 Dependencies

This document has the following dependencies

```
library(yamss)
library(mtbls2)
```

2 Processing a metabolomics dataset

We will be looking at data provided in the mtbls2 data package. In this experiment, investigators exposed wild-type and mutant *Arabidopsis thaliana* leaves to silver nitrate and performed global LC/MS profiling. The experiment was repeated twice, but we will only be looking at the first replicate. There are 4 wild-type and 4 mutant plants in this experiment.

```r
filepath <- file.path(find.package("mtbls2"), "mzData")
files <- list.files(filepath, pattern = "MSpos-Ex1", recursive = TRUE, full.names = TRUE)
classes <- rep(c("wild-type", "mutant"), each = 4)
```

The first step is to read the raw mass spec data into an R representation using `readMSdata()`:

```r
colData <- DataFrame(sampClasses = classes, ionmode = "pos")
cmsRaw <- readMSdata(files = files, colData = colData, mzsubset = c(500,520), verbose = TRUE)
```

The output of `readMSdata()` is an object of class CMSraw representing raw (unprocessed) data. We use the `colData` argument to store phenotype information about the different samples.

The next step is to use `bakedpi()` to preprocess these samples. This function takes a while to run, so we only run it on a small slice of M/Z values, using the `mzsubset` argument. This is only done for speed.

```r
cmsProc <- bakedpi(cmsRaw, dbandwidth = c(0.01, 10), dgridstep = c(0.01, 1),
                   outfileDens = NULL, dortalign = TRUE, mzsubset = c(500, 520), verbose = TRUE)
```

The `dbandwidth` and `dgridstep` arguments influence the bivariate kernel density estimation which forms the core of `bakedpi`. `dgridstep` is a vector of length 2 that specifies the spacing of the grid upon which the density is estimated. The first component specifies the spacing in the M/Z direction, and the second component specifies the spacing in the scan (retention time) direction. To showcase a fast example, we have specified the M/Z and scan spacings to be 0.01 and 1 respectively, but we recommend keeping the defaults of 0.005 and 1 because this will more accurately define the M/Z and scan bounds of the detected peaks. `dbandwidth` is a vector of length 2 that specifies the kernel density bandwidth in the M/Z and scan directions in the first and second components respectively. Note that `dbandwidth[1]` should be greater than or equal to `dgridstep[1]` and `dbandwidth[2]` should be greater than or equal to `dgridstep[2]`. Because a binning strategy is used to speed up computation of the density estimate, this is to ensure that data points falling into the same grid location all have the same distances associated with them.
For experiments with a wide range of M/Z values or several thousands of scans, the computation of the density estimate can be time-intensive. For this reason, it can be useful to save the density estimate in an external file specified by the outfileDens argument. If outfileDens is set to NULL, then the density estimate is not saved and must be recomputed if we wish to process the data again. Specifying the filename of the saved density estimate in outfileDens when rerunning bakedpi() skips the density estimation phase which can save a lot of time.

The resulting object is an instance of class CMSproc which contains the bivariate kernel density estimate as well as some useful preprocessing metadata. In order to obtain peak bounds and quantifications, the last step is to run slicepi(), which computes a global threshold for the density, uses this threshold to call peak bounds, and quantifies the peaks. If the cutoff argument is supplied as a number, then slicepi() will use this as the density threshold. Otherwise if cutoff is left as the default NULL, a data-driven threshold will be identified.

```r
cmsSlice <- slicepi(cmsProc, cutoff = NULL, verbose = TRUE)
## [slicepi] Computing cutoff
## [slicepi] Computing peak bounds
## [slicepi] Quantifying peaks
cmsSlice
## An object of class 'CMSslice'
## Representing 8 data files
## Number of scans: 3389
## M/Z: 500.000120 - 519.998100
## Number of peaks: 72
```

The output of slicepi() is an instance of class CMSslice and contains the peak bounds and quantifications as well as sample and preprocessing metadata.

### 3 Differential Analysis

We can access the differential analysis report with diffrep(). This is a convenience function, similar to diffreport() from the xcms package. In our case it uses limma to do differential analysis; the output of diffrep() is basically topTable() from limma.

```r
dr <- diffrep(cmsSlice, classes = classes)
head(dr)
## logFC AveExpr t P.Value adj.P.Val B
## 25  3.474403 15.32845  70.51488  9.110811e-12  3.650248e-10  17.95487
## 54 -2.156827  17.52813 -69.50470  1.013958e-11  3.650248e-10  17.84146
## 21  1.209102  17.87641  44.72779  2.655826e-10  4.527970e-09  14.29968
## 55 -2.004350  15.66532 -43.91037  3.044188e-10  4.527970e-09  14.14912
## 4  -1.137775  16.23114 -43.71853  3.144424e-10  4.527970e-09  14.11336
## 47  1.252761  17.88478  35.34145  1.515174e-09  1.818209e-08  12.36901
```

Let's look at the peak bound information for the peaks with the strongest differential signal. We can store the IDs for the top 10 peaks with

```r
topPeaks <- as.numeric(rownames(dr)[1:10])
topPeaks
## [1] 25 54 21 55 4 47 46 16 59 17
```

We can access the peak bound information with peakBounds() and select the rows corresponding to topPeaks.

```r
bounds <- peakBounds(cmsSlice)
idx <- match(topPeaks, bounds[,"peaknum")
bounds[idx,]
## DataFrame with 10 rows and 5 columns
## mzmin mzmax scanmin scanmax peaknum
## 4 Information contained in a CMSproc object

CMSproc objects contain information useful in exploring your data.

### 4.1 Density estimate

The bivariate kernel density estimate matrix can be accessed with `densityEstimate()`.

```r
dmat <- densityEstimate(cmsProc)
```

We can make a plot of the density estimate in a particular M/Z and scan region with `plotDensityRegion()`.

```r
mzrange <- c(bounds[idx[1], "mzmin"], bounds[idx[1], "mzmax"])
scanrange <- c(bounds[idx[1], "scanmin"], bounds[idx[1], "scanmax"])
plotDensityRegion(cmsProc, mzrange = mzrange + c(-0.5,0.5), scanrange = scanrange + c(-30,30))
```
Peaks are called by thresholding the density estimate. If we wish to investigate the impact of varying this cutoff, we can use `densityCutoff` to obtain the original cutoff and `updatePeaks()` to re-call peaks and quantify them. Quantiles of the non-zero density values are also available via `densityQuantiles()`.

```r
cmsSlice2 <- sliceπ(cmsProc, densityCutoff(cmsSlice) * 0.99)
  ## [sliceπ] Computing peak bounds
  ## [sliceπ] Quantifying peaks
dqs <- densityQuantiles(cmsProc)
  head(dqs)
  ## 0.0% 0.1% 0.2% 0.3% 0.4%
  ## 2.270341e-22 8.066129e-20 1.616783e-19 2.473895e-19 3.376696e-19
  ## 0.5%
  ## 4.398691e-19
cmsSlice3 <- sliceπ(cmsProc, dqs["98.5%"])
  ## [sliceπ] Computing peak bounds
  ## [sliceπ] Quantifying peaks
nrow(peakBounds(cmsSlice)) # Number of peaks detected - original
  ## [1] 72
nrow(peakBounds(cmsSlice2)) # Number of peaks detected - updated
  ## [1] 72
nrow(peakBounds(cmsSlice3)) # Number of peaks detected - updated
  ## [1] 206
```
5 Sessioninfo

- R version 3.3.1 (2016-06-21), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, BiocStyle 2.2.0, GenomicInfoDb 1.10.0, GenomicRanges 1.26.0, IRanges 2.8.0, S4Vectors 0.12.0, SummarizedExperiment 1.4.0, mtbls2 1.3.2, yamss 1.0.0
- Loaded via a namespace (and not attached): EBImage 4.16.0, Matrix 1.2-7.1, ProtGenerics 1.6.0, Rcpp 0.12.7, XVector 0.14.0, abind 1.4-5, assertthat 0.1, chron 2.3-47, codetools 0.2-15, data.table 1.9.6, digest 0.6.10, evaluate 0.10, fftwtools 0.9-7, formatR 1.4, grid 3.3.1, htmltools 0.3.5, jpeg 0.1-8, knitr 1.14, lattice 3.20-34, limma 3.30.0, locfit 1.5-9.1, magrittr 1.5, mzR 2.8.0, png 0.1-7, rmarkdown 1.1, stringi 1.1.2, stringr 1.1.0, tibble 1.2, tiff 0.1-5, tools 3.3.1, yam 2.1.13, zlibbioc 1.20.0

References
