Cardinal: Analytic tools for mass spectrometry imaging

Kyle D. Bemis

July 23, 2014

Contents

1 Introduction **1**

2 Setup **1**

3 Input/Output **2**
3.1 Input . . . 2
3.1.1 Analyze 7.5 . . . 2
3.1.2 imzML . . . 2
3.2 Output . . . 2
3.2.1 RData files . . . 2

4 Subsetting and Inspecting Data **2**

5 Plotting **3**
5.1 Formula interface . . . 3
5.2 Plotting using *Cardinal* . . . 3
5.2.1 Plotting mass spectra . . . 4
5.2.2 Plotting ion images . . . 4

6 Pre-processing **6**
6.1 Normalization . . . 7
6.2 Smoothing . . . 7
6.3 Baseline reduction . . . 8
6.4 Peak picking . . . 8
6.5 Peak alignment . . . 9
6.6 Data reduction . . . 9

7 Analysis **10**

8 Advanced Topics **10**
8.1 Apply . . . 10
8.1.1 `pixelApply` . . . 11
8.1.2 `featureApply` . . . 13
8.2 Simulation . . . 14

9 Session info **14**

1 Introduction

This will be a brief walkthrough of some of the basic functionality of Cardinal.

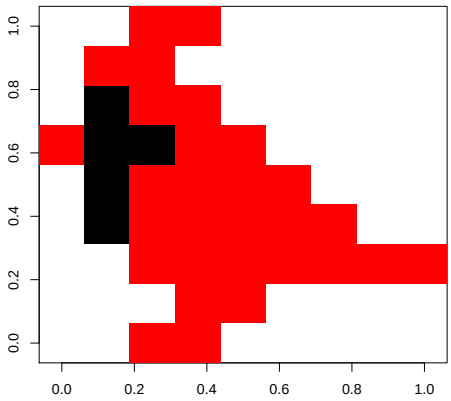

Figure 1: Ground truth image used to generate the simulated dataset.

2 Setup

For the following examples, we will use a simulated dataset. The image is a cardinal with red and black feathers, where the colors represent different regions of the image. The mass spectra will have two peaks to indicate the two regions. We use `generateImage` to generate the dataset from an integer matrix where 0 represents black regions of the image and 1 represents the red regions of the image.

```
> data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
+   NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
+   1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
+   1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
+   1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)
```

We can plot the ground truth image directly.

```
> image(data[,ncol(data):1], col=c("black", "red"))
```

Now we generate the data as if from a mass spectrometry imaging experiment with peaks at m/z 3000 (higher intensity in black pixels) and m/z 4000 (higher intensity in red pixels).

```
> set.seed(1)
> msset <- generateImage(data, range=c(1000,5000), centers=c(3000,4000), resolution=100,
+          step=3.3, as="MSImageSet")
```

We need to mark which pixels are black and which are red.

```
> pData(msset)$pg <- factor(data[is.finite(data)], labels=c("black", "red"))
```

Then we need to mark which features (which regions of the mass spectrum) belong to the peaks associated with "black" or "red" pixels; the rest of the spectrum is marked as background noise (bg).

```
> fData(msset)$fg <- factor(rep("bg", nrow(fData(msset))), levels=c("bg", "black", "red"))
> fData(msset)$fg[2950 < fData(msset)$mz & fData(msset)$mz < 3050] <- "black"
> fData(msset)$fg[3950 < fData(msset)$mz & fData(msset)$mz < 4050] <- "red"
```

Now we can experiment with different ways of working with a mass spectrometry imaging dataset in *Cardinal*.

3 Input/Output

3.1 Input

Cardinal can read two of the most common data exchange formats in imaging mass spectrometry: Analyze 7.5 and imzML.

3.1.1 Analyze 7.5

Originally designed for MRI use, Analyze 7.5 is one of the oldest and most common formats still used for exchange of mass spectrometry imaging data.

```
> name <- "Bierbaum_demo_"
> folder <- "/Users/kuwisdelu/Documents/Datasets/DESI-Imaging/Bierbaum_demo_"
> x1 <- readAnalyze(name=name, folder=folder)
```

3.1.2 imzML

A newly-developed format specifically designed for interchange of mass spectrometry imaging datasets, imzML is an open XML-based format to which many other formats can be converted.

```
> name <- "S042_Continuous"
> folder <- "/Users/kuwisdelu/Documents/Purdue/Research/Imaging/Data and Code/imzML/s042_continuous/"
> x2 <- readImzML(name=name, folder=folder)
```

3.2 Output

3.2.1 RData files

Any R object including `MSImageSet` datasets can be exported and saved as an **RData** file using `save` and reimported using `load`.

4 Subsetting and Inspecting Data

5 Plotting

One of the most important parts of working with mass spectrometry imaging datasets is visualization of the data by examining the ion images and mass spectra. *Cardinal* provides powerful functionality for plotting both ion images, mass spectra, as well as other representations of imaging data.

5.1 Formula interface

The plotting facilities of *Cardinal* are based on the powerful formula interface used by the `lattice` graphics package.

5.2 Plotting using Cardinal

For the following examples, we will use a simulated dataset. The image is a cardinal with red and black feathers, where the colors represent different regions of the image. The mass spectra will have two peaks to indicate the two regions. We use `generateImage` to generate the dataset from an integer matrix where 0 represents black regions of the image and 1 represents the red regions of the image.

```
> data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, 1, 1, NA, NA,
+   NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
+   1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
```

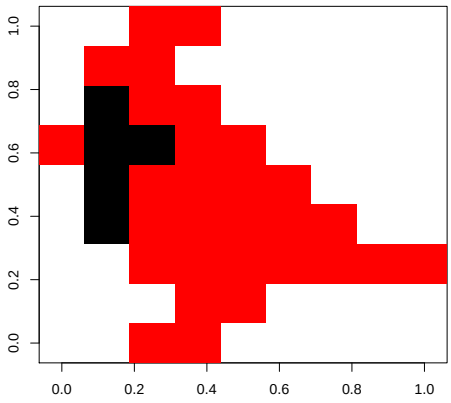

Figure 2: Ground truth image used to generate the simulated dataset.

```
+  1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
+  1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)
```

We can plot the ground truth image directly.

```
> image(data[,ncol(data):1], col=c("black", "red"))
```

Now we generate the data as if from a mass spectrometry imaging experiment with peaks at m/z 3000 (higher intensity in black pixels) and m/z 4000 (higher intensity in red pixels).

```
> set.seed(1)
> msset <- generateImage(data, range=c(1000,5000), centers=c(3000,4000), resolution=100,
+   step=3.3, as="MSImageSet")
```

We need to mark which pixels are black and which are red.

```
> pData(msset)$pg <- factor(data[is.finite(data)], labels=c("black", "red"))
```

Then we need to mark which features (which regions of the mass spectrum) belong to the peaks associated with "black" or "red" pixels; the rest of the spectrum is marked as background noise (bg).

```
> fData(msset)$fg <- factor(rep("bg", nrow(fData(msset))), levels=c("bg", "black", "red"))
> fData(msset)$fg[2950 < fData(msset)$mz & fData(msset)$mz < 3050] <- "black"
> fData(msset)$fg[3950 < fData(msset)$mz & fData(msset)$mz < 4050] <- "red"
```

Now we can experiment with different ways of plotting an imaging dataset.

5.2.1 Plotting mass spectra

The `plot` method is used to plot mass spectra. The `pixel` argument is used to specify the pixel to use to plot the mass spectrum. If no conditioning is desired, the formula does not need to be specified explicitly.

```
> plot(msset, pixel=1)
```

```
> plot(msset, ~ mz, pixel=1)
```

Specifying multiple pixels will apply a function, specified by `fun`, over those pixels. This can be used to create a plot of the mean spectrum (the default behavior). Below we obtain the mean spectrum of the red pixels, and the max spectrum of the black pixels.

```
> plot(msset, pixel=pData(msset)$pg=="red", fun=median, main="Median of red pixels")
```

```
> plot(msset, pixel=pData(msset)$pg=="black", fun=max, main="Max of black pixels")
```

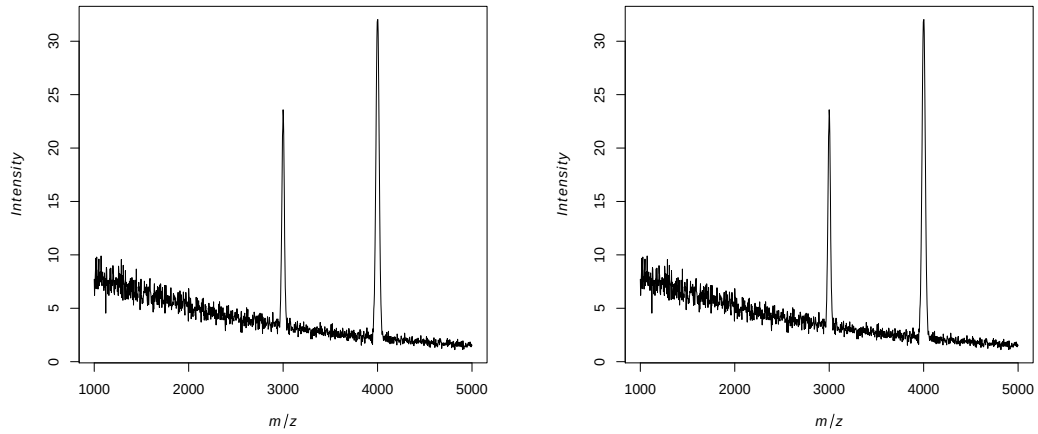

Figure 3: A simple mass spectrum plot. Both forms produce the same plot.

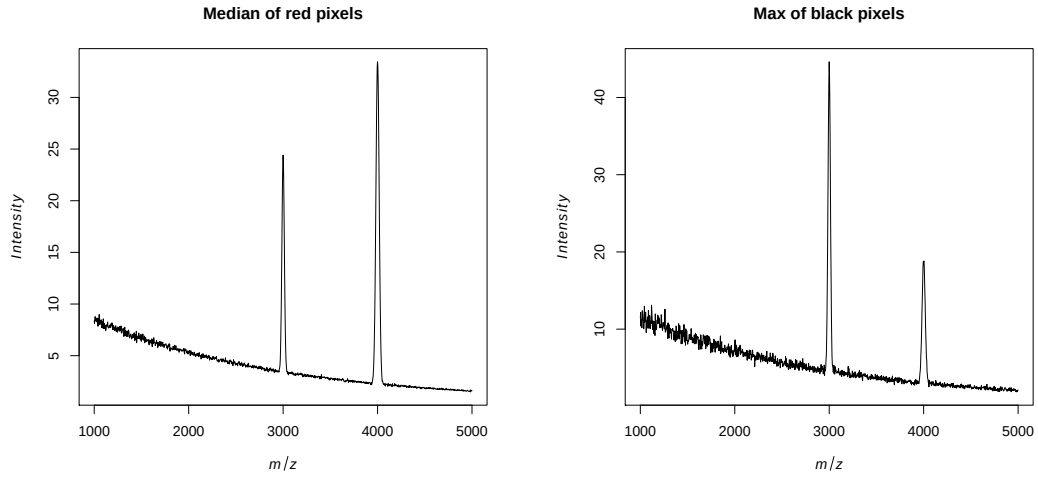

Figure 4: Applying a function over pixels to plot a median and max spectrum.

Using the `lattice` graphics option allows for more complex plots to be made. Conditioning on variables in the `formula` argument allows direct comparison between regions of the image or mass spectrum. For example, by conditioning on the variable `pData(msset)$pg` which specifies the color of the pixels, we can obtain mean spectra for each type of pixel in a single step; notice that the `plot` method knows where to find the `pg` variable, because it is contained in `msset`. Likewise, we use the `fg` variable (which we used to mark notable m/z-values) with the argument `groups` to distinguish different regions of the mass spectrum with different colors.

```
> print(plot(msset, ~ mz | pg, pixel=1:ncol(msset), groups=fg, lattice=TRUE, col=c("blue", "black", "re
```

5.2.2 Plotting ion images

The `image` method is used to plot images. The `feature` argument is used to specify the feature to use to create the image. For a mass spectrometry imaging dataset, the features are the m/z-values corresponding to single-ion images. As before, if no conditioning is desired, the formula does not need to be specified explicitly.

```
> image(msset, feature=1, col.regions=gradient.colors(100, "red", "black"))
```

```
> image(msset, ~ x * y, feature=1, col.regions=gradient.colors(100, "red", "black"))
```

Like with the `plot` method, the `image` method can apply functions over features (m/z-values) when multiple features are specified. By default, `mean` is used to average the images over the features. In the following example, we specify

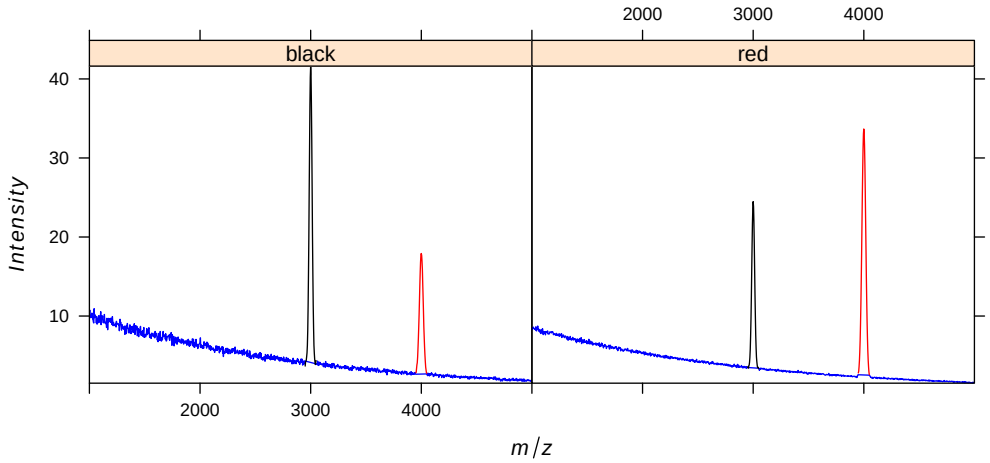

Figure 5: A plot conditioning on variables using `lattice` graphics.

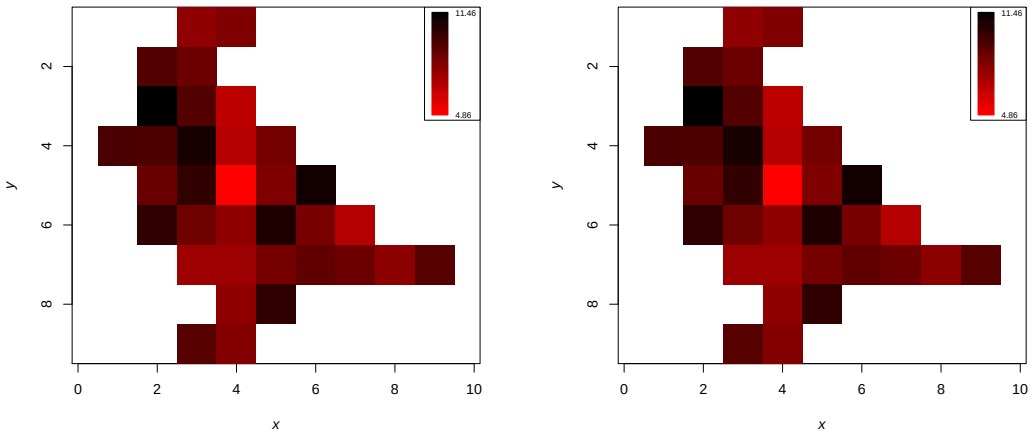

Figure 6: A simple single-ion image. Both forms produce the same plot.

two plots, first using the features from the peak that has a higher intensity associated with black pixels, and then using the features from the peak that has a higher intensity associated with red pixels.

```
> image(msset, feature=fData(msset)$fg=="black", col.regions=alpha.colors(100, "black"))
```

```
> image(msset, feature=fData(msset)$fg=="red", col.regions=alpha.colors(100, "red"))
```

Using a `lattice`-style formula, we can condition on other variables with `image` too. Here we use all of the features, but condition on which part of the mass spectrum those features come from using the variable `fData(msset)$fg`. Again, since `image` knows to look in `msset`, we only need to specify the variable as `fg`.

```
> print(image(msset, ~ x * y | fg, feature=1:nrow(msset), col.regions=intensity.colors(100), lattice=TR
```

6 Pre-processing

6.1 Normalization

Normalization is perhaps the most important pre-processing step before any kind of analysis should be performed on biological datasets, and mass spectrometry imaging experiments are no different in this regard. *Cardinal* provides

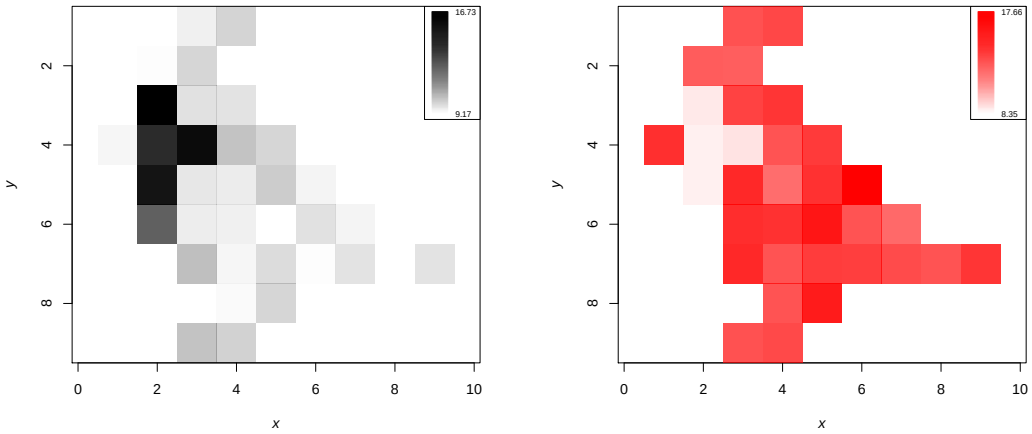

Figure 7: Averaging over different sets of mass features.

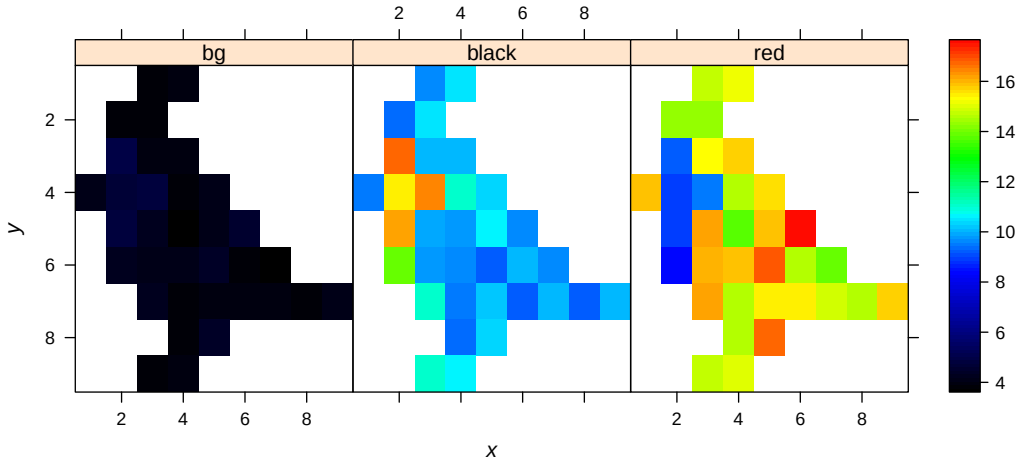

Figure 8: Images conditioning on variables using `lattice` graphics.

normalization to total ion current (TIC). In the first command below, we only perform the normalization on the first pixel in order to show a plot of the processing results. In the second, we perform normalization on the whole dataset.

```
> temp <- normalize(msset, pixel=1, method="tic", plot=TRUE)
> msset2 <- normalize(msset, method="tic")
```

6.2 Smoothing

Smoothing the mass spectra is useful for removing noise, which can improve detection of peaks. *Cardinal* provides several common methods for smoothing mass spectra, including Gaussian kernel smoothing, Savitsky-Golay smoothing, and a simple moving average filter.

```
> temp <- smoothSignal(msset2, pixel=1, method="gaussian", window=9, plot=TRUE)
> temp <- smoothSignal(msset2, pixel=1, method="sgolay", window=15, plot=TRUE)
> msset3 <- smoothSignal(msset2, method="gaussian", window=9)
```

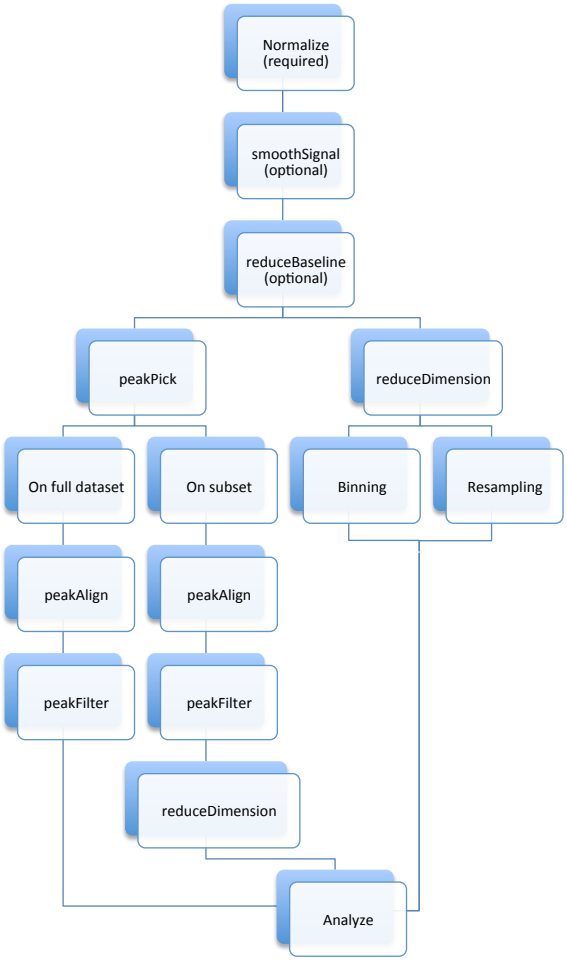

Figure 9: Preprocessing steps

6.3 Baseline reduction

Baseline reduction is often necessary for many datasets, and *Cardinal* implements a simple version that interpolates a baseline from local medians or local minima, while attempting to preserve the signal from mass spectral peaks.

```
> temp <- reduceBaseline(msset3, pixel=1, method="median", blocks=50, plot=TRUE)
```

```
> msset4 <- reduceBaseline(msset3, method="median", blocks=50)
```

6.4 Peak picking

Peak picking is a common form of data reduction that reduces the signal to relevant data peaks. *Cardinal* implements three varieties based on a user-specified signal-to-noise ratio (SNR). The "simple" version interpolates a constant noise pattern, the "adaptive" version interpolates an adaptive noise pattern, and "limpic" implements the LIMPIC algorithm for peak detection.

```
> temp <- peakPick(msset4, pixel=1, method="adaptive", SNR=3, plot=TRUE)
```

```
> temp <- peakPick(msset4, pixel=1, method="limpic", SNR=3, plot=TRUE)
```

```
> msset5 <- peakPick(msset4, method="simple", SNR=3)
```

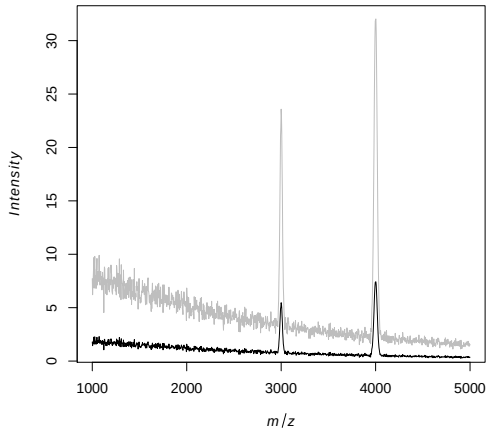

Figure 10: Total ion current (TIC) normalization.

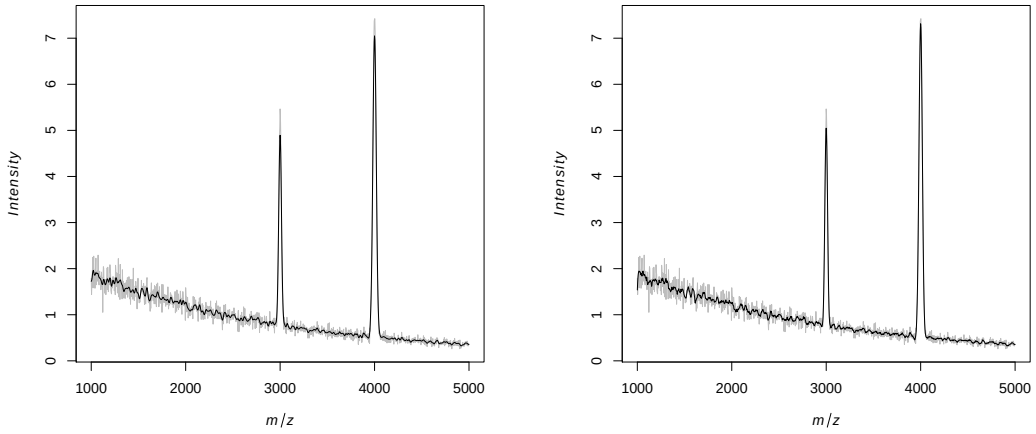

Figure 11: Gaussian smoothing and Savitsky-Golay smoothing.

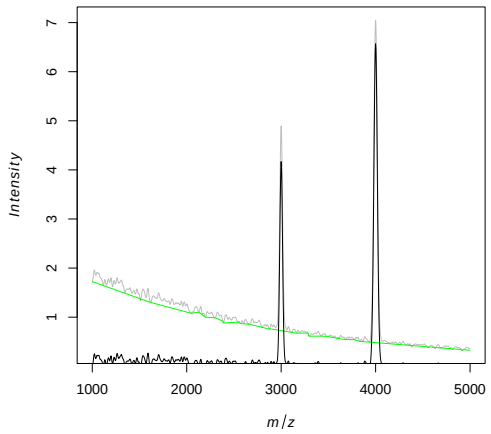

Figure 12: Baseline reduction using interpolation from medians.

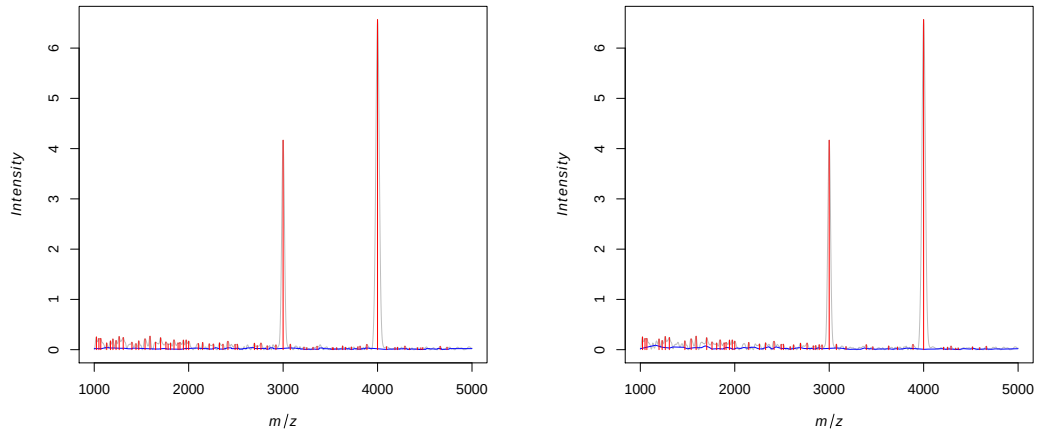

Figure 13: Peak picking with adaptive noise and LIMPIC.

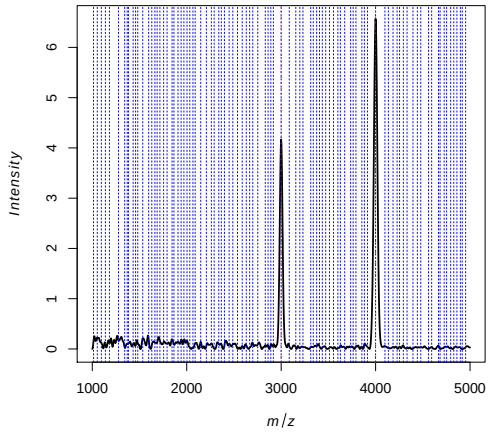

Figure 14: Peak alignment to the local maxima of the mean spectrum.

6.5 Peak alignment

Peak alignment is necessary to account for possible inaccuracy in m/z measurements. Peaks can be aligned to a reference list of known m/z values, or to the local maxima in the mean spectrum.

```
> temp <- peakAlign(msset5, pixel=1, method="diff", plot=TRUE)
```

```
> msset6 <- peakAlign(msset5, method="diff")
```

6.6 Data reduction

Other common forms of data reduction include resampling and binning.

```
> temp <- reduceDimension(msset4, pixel=1, method="bin", width=25, fun=mean, plot=TRUE)
```

```
> temp <- reduceDimension(msset4, pixel=1, method="resample", step=25, plot=TRUE)
```

```
> msset7 <- reduceDimension(msset4, method="resample", step=25)
```

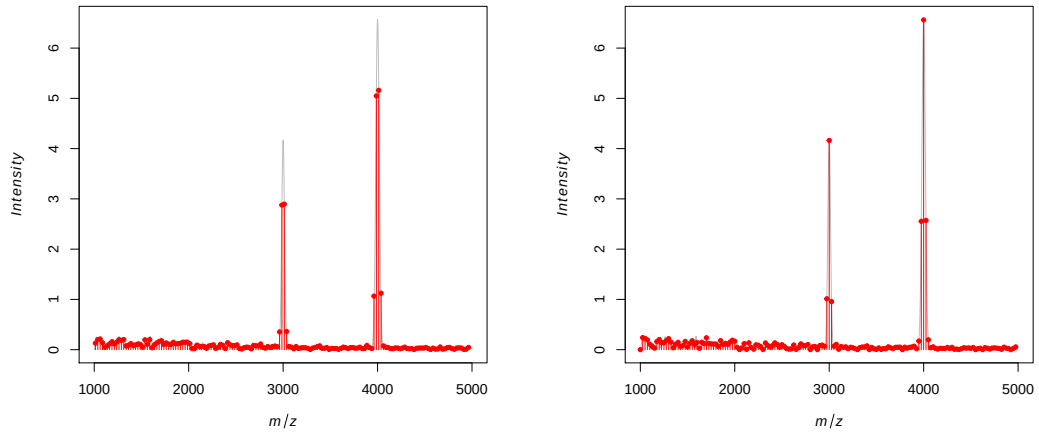

Figure 15: Data reduction via binning and resampling.

7 Analysis

8 Advanced Topics

8.1 Apply

The `apply` family of functions are a powerful feature of *R*. The `apply` function applies a function over margins of an array, while `sapply` applies a function over every element of a vector-like object. The function `tapply` applies a function over a "ragged" array, so that the function is applied over groups of values given by levels of another variable (usually a factor). In *Cardinal*, the methods `pixelApply` and `featureApply` allow `apply`-like functionality that combine traits of each of these, tailored for imaging datasets.

For the following examples, we will use a simulated dataset. The image is a cardinal with red and black feathers, where the colors represent different regions of the image. The mass spectra will have two peaks to indicate the two regions. We use `generateImage` to generate the dataset from an integer matrix where 0 represents black regions of the image and 1 represents the red regions of the image.

```
> data <- matrix(c(NA, NA, 1, 1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA,
+   NA, NA, NA, NA, NA, 0, 1, 1, NA, NA, NA, NA, NA, 1, 0, 0, 1,
+   1, NA, NA, NA, NA, NA, 0, 1, 1, 1, 1, NA, NA, NA, NA, 0, 1, 1,
+   1, 1, 1, NA, NA, NA, NA, 1, 1, 1, 1, 1, 1, 1, NA, NA, NA, 1,
+   1, NA, NA, NA, NA, NA, NA, 1, 1, NA, NA, NA, NA, NA), nrow=9, ncol=9)
```

We can plot the ground truth image directly.

```
> image(data[,ncol(data):1], col=c("black", "red"))
```

Now we generate the data as if from a mass spectrometry imaging experiment with peaks at m/z 3000 (higher intensity in black pixels) and m/z 4000 (higher intensity in red pixels).

```
> set.seed(1)
> msset <- generateImage(data, range=c(1000,5000), centers=c(3000,4000), resolution=100,
+   step=3.3, as="MSImageSet")
```

We need to mark which pixels are black and which are red.

```
> pData(msset)$pg <- factor(data[is.finite(data)], labels=c("black", "red"))
```

Then we need to mark which features (which regions of the mass spectrum) belong to the peaks associated with "black" or "red" pixels; the rest of the spectrum is marked as background noise (`bg`).

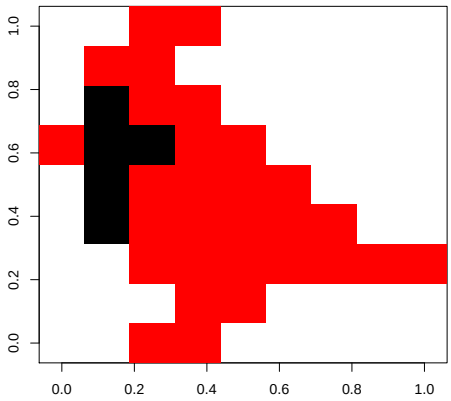

Figure 16: Ground truth image used to generate the simulated dataset.

```
> fData(msset)$fg <- factor(rep("bg", nrow(fData(msset))), levels=c("bg", "black", "red"))
> fData(msset)$fg[2950 < fData(msset)$mz & fData(msset)$mz < 3050] <- "black"
> fData(msset)$fg[3950 < fData(msset)$mz & fData(msset)$mz < 4050] <- "red"
```

Now we can experiment with different ways of plotting an imaging dataset.

8.1.1 pixelApply

The method `pixelApply` allows functions to be applied over all pixels. The function is applied pixel-by-pixel to the feature vectors (mass spectra). Here, we use `pixelApply` to find the pixel-by-pixel mean intensity of different regions of the mass spectrum. We provide `fData(msset)$fg` as a grouping variable, since it indicates different regions of the mass spectrum we expect to be associated with either background noise, or red or black pixels. Since `pixelApply` knows to look in `msset` for the variable, we only need to provide `fg` to the argument `.feature.groups`.

```
> p1 <- pixelApply(msset, mean, .feature.groups=fg)
> p1[,1:30]
```

```
      x = 3, y = 1 x = 4, y = 1 x = 2, y = 2 x = 3, y = 2
bg        3.890303     3.935974     3.852787     3.789505
black     9.635005    10.517746     9.308453    10.440883
red      14.765521    15.123040    14.341188    14.303296
      x = 2, y = 3 x = 3, y = 3 x = 4, y = 3 x = 1, y = 4
bg        4.976931     3.960794     4.028191     4.145899
black    16.728553    10.093288    10.064517     9.532089
red       9.241133    15.285169    15.723496    15.951808
      x = 2, y = 4 x = 3, y = 4 x = 4, y = 4 x = 5, y = 4
bg        4.632393     4.863252     3.813765     4.065992
black    15.457280    16.399439    11.023859    10.385429
red       8.928271     9.437796    14.641915    15.559322
      x = 2, y = 5 x = 3, y = 5 x = 4, y = 5 x = 5, y = 5
bg        4.853165     4.189047     3.613069     4.109948
black    16.177474     9.967317     9.820366    10.696623
red       8.958284    16.241629    13.738381    15.884999
      x = 6, y = 5 x = 2, y = 6 x = 3, y = 6 x = 4, y = 6
bg        4.516140     4.230802     4.146095     4.120097
black     9.599486    13.888708     9.756209     9.680859
red      17.661532     8.354190    16.029427    15.867799
      x = 5, y = 6 x = 6, y = 6 x = 7, y = 6 x = 3, y = 7
bg        4.334037     3.876803     3.702975     4.190092
```

```
black     9.174036    10.088559     9.622180    11.097352
red      16.825779    14.652355    13.854740    16.202813
      x = 4, y = 7 x = 5, y = 7 x = 6, y = 7 x = 7, y = 7
bg        3.848391     4.028779     4.018565     3.932754
black     9.506104    10.272061     9.273227    10.025520
red      14.646689    15.464322    15.394325    14.869473
      x = 8, y = 7 x = 9, y = 7
bg        3.855238     4.112314
black     9.227148    10.014673
red      14.623218    15.741579
```

By comparing side-by-side with the ground truth (which we have stored in the variable `pData(msset)$pg`), we see the result is as we expected. For "black" pixels, the mean intensity of features belonging to the "black"-associated peak (m/z 3000) is higher, while for the "red" pixels, the mean intensity of features belonging to the "red"-associated peak (m/z 4000) is higher.

```
> cbind(pData(msset), t(p1))[1:30,c("pg", "black", "red")]
```

```
                pg     black       red
x = 3, y = 1   red  9.635005 14.765521
x = 4, y = 1   red 10.517746 15.123040
x = 2, y = 2   red  9.308453 14.341188
x = 3, y = 2   red 10.440883 14.303296
x = 2, y = 3 black 16.728553  9.241133
x = 3, y = 3   red 10.093288 15.285169
x = 4, y = 3   red 10.064517 15.723496
x = 1, y = 4   red  9.532089 15.951808
x = 2, y = 4 black 15.457280  8.928271
x = 3, y = 4 black 16.399439  9.437796
x = 4, y = 4   red 11.023859 14.641915
x = 5, y = 4   red 10.385429 15.559322
x = 2, y = 5 black 16.177474  8.958284
x = 3, y = 5   red  9.967317 16.241629
x = 4, y = 5   red  9.820366 13.738381
x = 5, y = 5   red 10.696623 15.884999
x = 6, y = 5   red  9.599486 17.661532
x = 2, y = 6 black 13.888708  8.354190
x = 3, y = 6   red  9.756209 16.029427
x = 4, y = 6   red  9.680859 15.867799
x = 5, y = 6   red  9.174036 16.825779
x = 6, y = 6   red 10.088559 14.652355
x = 7, y = 6   red  9.622180 13.854740
x = 3, y = 7   red 11.097352 16.202813
x = 4, y = 7   red  9.506104 14.646689
x = 5, y = 7   red 10.272061 15.464322
x = 6, y = 7   red  9.273227 15.394325
x = 7, y = 7   red 10.025520 14.869473
x = 8, y = 7   red  9.227148 14.623218
x = 9, y = 7   red 10.014673 15.741579
```

We can manually construct the images corresponding to the mean intensity of the two peaks centered at m/z 3000 and m/z 4000 and plot their images.

```
> temp1 <- MSImageSet(spectra=t(as.vector(p1["black",])), coord=coord(msset), mz=3000)
> image(temp1, feature=1, col=alpha.colors(100, "black"), main="black peak", sub="m/z = 3000")
```

```
> temp2 <- MSImageSet(spectra=t(as.vector(p1["red",])), coord=coord(msset), mz=4000)
> image(temp2, feature=1, col=alpha.colors(100, "red"), main="red peak", sub="m/z = 4000")
```

If only the plots are desired rather than the actual data, then `image` can be used to perform these steps automatically while producing the plot. See *Cardinal plotting* for how to do this.

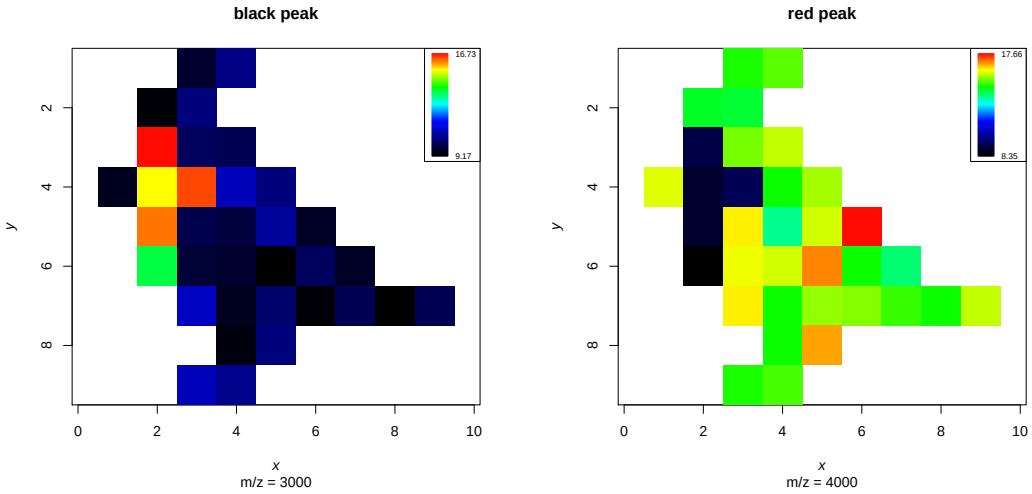

Figure 17: Mean intensites of the two peaks centered at m/z 3000 and m/z 4000.

8.1.2 featureApply

The method `featureApply` allows functions to be applied over all features. The function is applied to the flattened false-image vectors. The vectors are the pixel intensities of a single-feature image, disregarding missing pixels. Here, we use `featureApply` to find the mean spectrum for different groups of pixels. We provide `pData(msset)$pg` as a grouping variable, since it indicates the kind of pixel. We desire a mean spectrum for the black pixels and a mean spectrum for the red pixels. As before, since `featureApply` knows to look in `msset`, we only need to provide `pg` to the argument `.pixel.groups`.

```
> f1 <- featureApply(msset, mean, .pixel.groups=pg)
> f1[,1:30]
```

```
      m/z = 1000 m/z = 1003.3 m/z = 1006.6 m/z = 1009.9 m/z = 1013.2
black  10.098183    10.756344     9.784741    10.004682    10.066637
red     8.401082     8.289517     8.535487     8.463798     8.597804
      m/z = 1016.5 m/z = 1019.8 m/z = 1023.1 m/z = 1026.4
black     9.820493    10.270644     10.37729    10.046511
red       8.473853     8.407394      8.32738     8.363035
      m/z = 1029.7 m/z = 1033 m/z = 1036.3 m/z = 1039.6 m/z = 1042.9
black     9.583633   9.496959     9.212588     9.117748    10.933721
red       8.370147   8.233960     8.641274     8.144336     8.116496
      m/z = 1046.2 m/z = 1049.5 m/z = 1052.8 m/z = 1056.1
black     9.796224    10.161970    10.080725     9.962647
red       8.407537     8.498263     8.739433     8.305707
      m/z = 1059.4 m/z = 1062.7 m/z = 1066 m/z = 1069.3 m/z = 1072.6
black     9.891543     9.112999   9.941050     8.973674     9.650408
red       8.419085     8.085175   8.249085     7.999696     8.197752
      m/z = 1075.9 m/z = 1079.2 m/z = 1082.5 m/z = 1085.8
black     9.690155     9.353910     9.156658    10.054294
red       7.974962     7.785642     8.266194     8.041349
      m/z = 1089.1 m/z = 1092.4 m/z = 1095.7
black     9.232228     9.315825     9.696907
red       8.141642     7.832665     8.402787
```

Again, we can check the results by plotting them.

```
> plot(mz(msset), f1["black",], type="l", xlab="m/z", ylab="Intensity", main="mean spectrum of black pi
```

```
> plot(mz(msset), f1["red",], type="l", xlab="m/z", ylab="Intensity", main="mean spectrum of red pixels
```

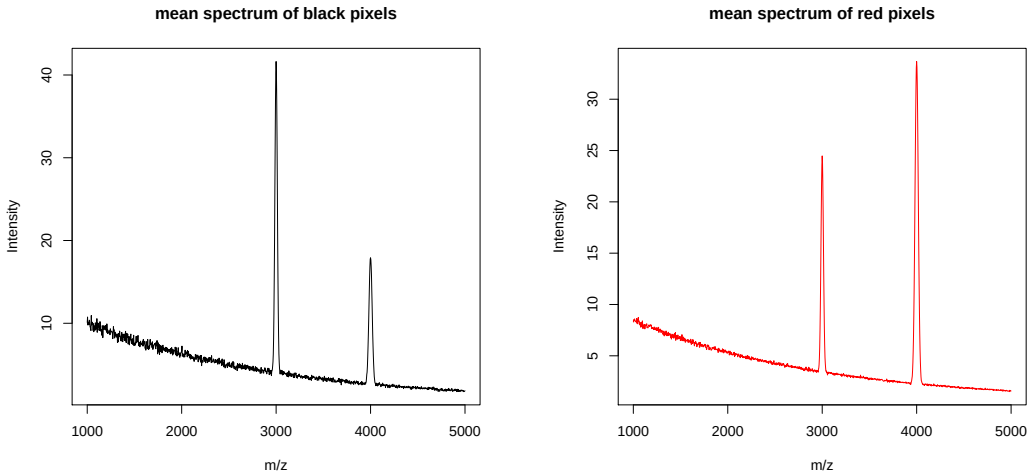

Figure 18: Mean intensites of the two peaks centered at m/z 3000 and m/z 4000.

As expected, we see the mean spectrum of the black pixels has a higher peak at m/z 3000 while the mean spectrum of the red pixels has a higher peak at m/z 4000. As before, if only the plots are desired rather than the actual data, then `plot` can be used to perform these steps automatically. See *Cardinal plotting* for how to do this.

8.2 Simulation

9 Session info

- R version 3.1.1 (2014-07-10), `x86_64-apple-darwin13.1.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: Biobase 2.24.0, BiocGenerics 0.10.0, Cardinal 0.8.2
- Loaded via a namespace (and not attached): BiocStyle 1.2.0, fields 7.1, grid 3.1.1, irlba 1.0.3, lattice 0.20-29, maps 2.3-7, MASS 7.3-33, Matrix 1.1-4, signal 0.7-4, sp 1.0-15, spam 0.41-0, stats4 3.1.1, tools 3.1.1