matter: Supplementary 2 - 3D mass spectrometry imaging case study

Kylie A. Bemis

October 22, 2021

Contents

1 Introduction

The first half of this vignette demonstrates the usefulness of *matter* for working with large mass spectrometry imaging (MSI) experiments in *Cardinal*. *Cardinal* is an R package for importing, pre-processing, visualization, and statistical analysis of mass spectrometry imaging experiments. *Cardinal* version $\geq$ 1.8 supports using *matter* as a backend for larger-than-memory datasets. More information is available at www.cardinalmsi.org.

The second half of this vignette presents an in-depth comparison in the performance between *matter*, *bigmemory*, and *ff* on real experimental datasets that are larger than memory.

2 Analyzing large 3D MSI experiments with *Cardinal* and *matter*

This example will use one of the benchmark 3D MSI experiments from Oetjen *et al.* [1]. We will use the 3D microbial timecourse experiment, which is comprised of interacting microbes at 3 time points, with a total of 17,672 pixels and 40,299 features. The data is stored in imzML format [2]. The ".imzML" XML file with experimental metadata is 30.5 MB, and the ".ibd" binary file with the m/z values and spectral intensities is 2.85 GB. This is one of the smaller datasets, making it a good place to begin.

Due to the various offsets in imzML ibd files, they cannot be attached as simply as *bigmemory* or *ff* files. These packages have strict requirements on the format of their data, for maximum computational effiency. *matter* takes a different approach with more flexibility, which allows use of imzML's domain-specific binary file format directly, and with minimal memory footprint, at the cost potentially slower computational performance in some situations.

```
> library(matter)
> library(Cardinal)
> path <- "~/Documents/Datasets/3D-MSI/3D_Timecourse/"
> file <- "Microbe_Interaction_3D_Timecourse_LP.imzML"
```

We load the dataset in *Cardinal* with the `readMSIData` function and the argument `attach.only=TRUE`. In *Cardinal* version $\geq$ 1.8, this automatically uses *matter*.

```
> msi <- readMSIData(paste0(path, file), attach.only=TRUE)
```

The data is attached as a `matter_mat` matrix. The matrix metadata takes up approximately 14.9 KB in memory, and points to 2.8 GB on disk.

The dataset was stored so that the first time point $t = 4$ corresponds to $z = 1, 2, 3, 4, 5, 6$, the second time point $t = 8$ corresponds to $z = 7, 8, 9, 10, 11, 12$, and the third time point $t = 11$ corresponds to $z = 13, 14, 15, 16, 17, 18$. We will reparamaterize the coordinates below to make it easier to work with the dataset. We also remove duplicated coordinates caused by converting the z-dimension to sequential integers.

```
> msi <- msi[,!duplicated(coord(msi))]
> msi$sample <- factor(sapply(msi$z, function(z) {
+         if ( z %in% 1:6 ) {
+                 1
+         } else if ( z %in% 7:12 ) {
+                 2
+         } else if ( z %in% 13:18 )  {
+                 3
+         }
+ }), labels=c("t = 4", "t = 8", "t = 11"))
> msi$z <- sapply(msi$z, function(z) {
+         if ( z %in% 1:6 ) {
+                 z
+         } else if ( z %in% 7:12 ) {
+                 z-6
+         } else if ( z %in% 13:18 )  {
+                 z-12
+         }
+ })
> msi$x <- mapply(function(x, t) {
+         switch(as.integer(t),
+                 x-30,
+                 x-15,
+                 x)
+ }, msi$x, msi$sample)
> varMetadata(msi)[c("x","y","z","sample"),"labelType"] <- "dim"
> protocolData(msi) <- AnnotatedDataFrame(
+         data=data.frame(row.names=sampleNames(msi)))
```

```
> msi <- regeneratePositions(msi)
> validObject(msi)
```

We can plot 3D molecular ion images using the `image3D` method.

```
> image3D(msi, ~ x * z * y, mz=262, theta=-55, contrast="suppress", layout=c(3,1))
```

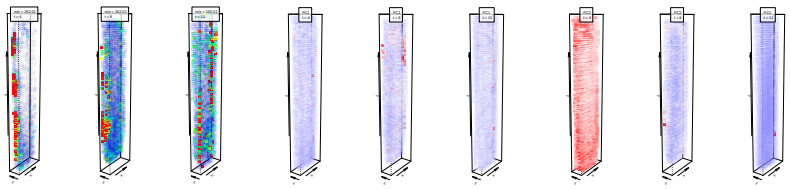
Now we perform principal components analysis using the `PCA` method of *Cardinal*.

```
> pca <- PCA(msi, ncomp=2, method="irlba", center=TRUE)
> pData(msi)[,c("PC1","PC2")] <- pca$scores[["ncomp = 2"]]
> fData(msi)[,c("PC1","PC2")] <- pca$loadings[["ncomp = 2"]]
```

We plot the first three principal components.

```
> image3D(msi, PC1 ~ x * z * y, theta=-55, col.regions=risk.colors(100), layout=c(3,1))
```

```
> image3D(msi, PC2 ~ x * z * y, theta=-55, col.regions=risk.colors(100), layout=c(3,1))
```



(a) m/z 262 **(b)** PC1 scores **(c)** PC2 scores

Figure 1: Plotting an ion image and the first 2 principal components for the 3D microbial time course experiment

3 Comparisons with alternative approaches

We will now illustrate the steps necessary for performing a principal components analysis using similar packages *bigmemory* or *ff*, and compare their performance with *matter*'s.

3.1 Using *bigmemory*

First we load *bigmemory* and create a blank `filebacked.big.matrix`. We will copy the data to this new matrix.

We must use *matter* to read the data and convert it to a `filebacked.big.matrix`, because the mass spectra are stored in a binary file incompatible with *bigmemory*.

Note that the original data elements are 32-bit floats:

```
> head(atomdata(iData(msi)))
```

However, if we want to use *bigalgebra* for matrix multiplication, we must change the data type from a 32-bit float to a 64-bit double. While *bigmemory* provides efficient matrix algebra routines in the *bigalgebra* package, they do not work with 32-bit floats; only 64-bit doubles are supported. (Note that *bigmemory* itself does support 32-bit float matrices; only *bigalgebra*'s native routines don't.)

```
> library(bigmemory)
> library(bigalgebra)
> backingfile <- paste0(expname, ".bin")
> backingpath <- "~/Documents/Temporary/"
> descriptorfile <- paste0(expname, ".desc")
> msi.bm <- filebacked.big.matrix(nrow=ncol(msi), ncol=nrow(msi),
+         backingfile=backingfile,
+         backingpath=backingpath,
+         descriptorfile=descriptorfile,
+         type="double")
```

Furthermore, we must transpose the matrix while converting it. This is because bioinformatics data is traditionally stored and manipulated using an P x N matrix, while most statistical functions in R expect a N x P matrix (where N is the number of samples and P is the number of features). *bigmemory* does not currently support virtually transposing a matrix, so it must be transposed now in order to perform PCA later.

```
> for ( i in seq_len(ncol(iData(msi))) )
+         msi.bm[i,] <- iData(msi)[,i]
```

Lastly, *bigmemory* does not support virtually scaling and centering rows or columns of a matrix. PCA should typically be performed on a centered data matrix. Although *bigmemory* supports arithmetic and algebra for `big.matrix` objects through the *bigalgebra* package, a new `big.matrix` with the transformed data is created as output.

When the file must already double in size (conversion from float to double) to accomodate matrix algebra, duplicating the matrix again simply to scale it seems an unacceptable waste of storage space.

Therefore, we implement implicit centering of the data matrix in a custom matrix multiplication function which can be passed to the `irlba` function.

```
> ct.mult.bm <- function(A, B, center = ct) {
+         if ( is.vector(A) ) {
+                 A <- t(A)
+                 cbind((A %*% B)[]) - (sum(A) * ct)
+         } else if ( is.vector(B) ) {
+                 B <- as.matrix(B)
+                 cbind((A %*% B)[]) - sum(B * ct)
+         }
+ }
```

Lastly, we calculate the mean of each column, and perform PCA using singular value decomposition via `irlba` with the custom multiplication function.

```
> library(biganalytics)
> library(irlba)
> ct <- apply(msi.bm, 2, mean)
```

```
> pca.out <- irlba(msi.bm, nu=0, nv=2, mult=ct.mult.bm)
> fData(msi)[,c("PC1","PC2")] <- pca.out$v
```

3.2 Using *ff*

We must again convert the dataset to a format compatible with *ff*. However, *ff* supports virtually transposing matrices, so we can keep the same virtual data layout as with *matter*.

While *ff* presents its own problems with matrix multiplication, they do not extend to the data type of the data elements, so we can keep the data as 32-bit floats, saving storage space as compared to *bigmemory*.

```
> library(ff)
> msi.ff <- ff(dim=c(nrow(msi), ncol(msi)),
+         filename=paste0(backingpath, backingfile),
+         vmode="single")
> for ( i in seq_len(ncol(iData(msi))) )
+                 msi.ff[,i] <- iData(msi)[,i]
> msi.ff <- vt(msi.ff)
```

However, the main *ff* package offers little in terms of arithmetic or algebraic operations that can be performed on `ff` objects. The *ffbase* package supplements and implements much of the arithmetic functionality missing from *ff*. However, it also creates brand new on-disk data files as the output rather than supporting virtual scaling and centering of matrices. Therefore, we again perform implicit centering during matrix multiplication using a custom matrix multiplication function.

Unfortunately, *ffbase* does not implement matrix multiplication for `ff` matrices. Fortunately, the package *bootSVD* provides a function that performs matrix multiplication of `ff` objects.

It is also worth noting that *ff*, *ffbase*, and *bootSVD* are each maintained by different developers.

```
> library(ffbase)
> library(bootSVD)
> ct.mult.ff <- function(A, B, center = ct) {
+         if ( is.vector(A) ) {
+                 A <- t(A)
+                 cbind(ffmatrixmult(A, B)[]) - (sum(A) * ct)
+         } else if ( is.vector(B) ) {
+                 B <- as.matrix(B)
+                 cbind(ffmatrixmult(A, B)[]) - sum(B * ct)
+         }
+ }
```

Now we can calculate the mean of each column, and perform PCA using singular value decomposition via `irlba` with the custom multiplication function.

```
> ct <- as.vector(ffapply(X=msi.ff, MARGIN=2, AFUN=mean, RETURN=TRUE)[])
> pca.out <- irlba(msi.ff, nu=0, nv=2, mult=ct.mult.ff)
> fData(msi)[,c("PC1","PC2")] <- pca.out$v
```

3.3 Evaluating performance between approaches

Principal components analysis					
Dataset	Size	Method	Mem. used	Mem. overhead	Time
3D Microbial Time Course	2.9 GB	matter	228 MB	50 MB	13 min, 6 sec
		bigmemory	330 MB	141 MB	53 sec
		ff	278 MB	85 MB	19 min, 48 sec
3D Oral Squamous Cell Carcinoma	25.4 GB	matter	977 MB	668 MB	2 hr, 7 min, 9 sec
		bigmemory	408 MB	266 MB	2 hr, 28 min, 2 sec
		ff	–	–	–
3D Mouse Pancreas	26.4 GB	matter	628 MB	370 MB	2 hr, 12 min, 46 sec
		bigmemory	303 MB	164 MB	2 hr, 52 min, 10 sec
		ff	–	–	–
3D Mouse Kidney	41.8 GB	matter	1.5 GB	1074 MB	3 hr, 22 min, 23 sec
		bigmemory	617 MB	431 MB	4 hr, 29 min, 23 sec
		ff	–	–	–

Table 1: Performance comparison of *matter*, *bigmemory*, and *ff* in calculating the first two principal components of real datasets
Memory overhead is the maximum memory used during the execution minus the memory in use upon completion. Some cells are missing because the analysis could not be performed with *ff*.

Table 1 shows the amount of time and memory *matter*, *bigmemory*, and *ff* used when performing PCA on each of the 3D MSI datasets.

For three of the datasets, the number of the elements in the dataset exceeded that maximum size of an `ff` object. The total length of an `ff` object is stored as a 32-bit integer, so it can at most reference $2^{31} - 1$ data elements. Conversely, *matter* and *bigmemory* store the total length of an object as a 64-bit double, allowing up to 2^52 data elements.

While *bigmemory* was dramatically faster than either *matter* or *ff* for the smallest 2.8 GB dataset, for the larger datasets that actually exceeded available memory, *matter* was faster.

It should be noted that the memory consumption reported for *bigmemory* is likely erroneous. The analyses were timed using the `profmem` function of *matter*, which wraps R's basic garbage collector call. The memory use will therefore be accurate for objects that use R's garbage collector, but inaccurate for objects that do not. The *bigmemory* package uses `mmap` on Unix systems, which is not controlled by R. In fact, system memory consumption under *bigmemory* was dramatically more than *matter*, but the authors were not able to measure this from R.

For datasets which exceed available computer memory, *matter* appears to outperform both *bigmemory* and *ff*.

Additionally, working with these datasets in *bigmemory* and *ff* either required additional steps that were either not required with *matter*, or, for some datasets, were simply impossible.

Table 2 shows the amount of time and memory it took to convert the data to files compatible with *bigmemory* and *ff*. For *ff*, this was only possible for one dataset. The time for file conversion (not included in the timing from Table 1) represents a substantial proportion of the total time it would take to analyze these datasets. In addition, conversion to *bigmemory* required doubling the file size if matrix multiplication was to be performed using *bigalgebra*.

In summary, *matter* often allows working with datasets without file conversion, which is often preferred for the sake of reproducibility, and typically requires less effort even after file conversion.

File conversion					
Dataset	Size	Method	Mem. used	Mem. overhead	Time
3D Microbial Time Course	2.9 GB	bigmemory	235 MB	46 MB	7 min, 12 sec
		ff	224 MB	32 MB	1 min, 16 sec
3D Oral Squamous Cell Carcinoma	25.4 GB	bigmemory	1.4 GB	954 MB	51 min, 3 sec
		ff	–	–	–
3D Mouse Pancreas	26.4 GB	bigmemory	630 MB	256 MB	1 hr, 31 min, 30 sec
		ff	–	–	–
3D Mouse Kidney	41.8 GB	bigmemory	1.5 GB	781 MB	1 hr, 47 min, 58 sec
		ff	–	–	–

Table 2: Time and memory used converting the dataset to a file compatible with *bigmemory* and/or *ff*
Memory overhead is the maximum memory used during the execution minus the memory in use upon completion. Some cells are missing because the conversion could not be performed for *ff*.

4 An R script for comparing performance

```
> library(matter)
> datapath <- "~/Documents/Datasets/3D-MSI/"
```

```
> require(Cardinal)
> # file <- "3D_Timecourse/Microbe_Interaction_3D_Timecourse_LP.imzML"
> # expname <- "3D_Timecourse"
>
> # file <- "3D_OSCC/3D_OSCC.imzML"
> # expname <- "3D_OSCC"
>
> # file <- "3D_Mouse_Pancreas/3D_Mouse_Pancreas.imzML"
> # expname <- "3D_Mouse_Pancreas"
>
> file <- "3DMouseKidney/3DMouseKidney.imzML"
> expname <- "3DMouseKidney"
> msi <- readMSIData(paste0(datapath, file), attach.only=TRUE)
```

```
> msi.prof[[expname]][["matter"]] <- profmem({
+
+         pca.out <- PCA(msi, ncomp=2, method="irlba", center=TRUE)
+         pData(msi)[,c("PC1","PC2")] <- pca.out$scores[[1]]
+         fData(msi)[,c("PC1","PC2")] <- pca.out$loadings[[1]]
+
+ })
```

```
> require(bigmemory)
> require(bigalgebra)
> backingfile <- paste0(expname, ".bin")
> backingpath <- "~/Documents/Temporary/"
> descriptorfile <- paste0(expname, ".desc")
> msi.bm <- filebacked.big.matrix(nrow=ncol(msi), ncol=nrow(msi),
+         backingfile=backingfile,
```

```
+         backingpath=backingpath,
+         descriptorfile=descriptorfile,
+         type="double")
> msi.prof[[expname]][["convert.bigmemory"]] <- profmem({
+
+         for ( i in seq_len(ncol(iData(msi))) )
+                 msi.bm[i,] <- iData(msi)[,i]
+
+ })
```

```
> ct.mult.bm <- function(A, B, center = ct) {
+         if ( is.vector(A) ) {
+                 A <- t(A)
+                 cbind((A %*% B)[]) - (sum(A) * ct)
+         } else if ( is.vector(B) ) {
+                 B <- as.matrix(B)
+                 cbind((A %*% B)[]) - sum(B * ct)
+         }
+ }
```

```
> require(biganalytics)
> require(irlba)
> msi.prof[[expname]][["bigmemory"]] <- profmem({
+
+         ct <- apply(msi.bm, 2, mean)
+
+         pca.out <- irlba(msi.bm, nu=0, nv=2, mult=ct.mult.bm)
+
+ })
> file.remove(paste0(backingpath, backingfile))
```

```
> require(ff)
> library(ffbase)
> require(bootSVD)
> msi.ff <- ff(dim=c(nrow(msi), ncol(msi)),
+         filename=paste0(backingpath, backingfile),
+         vmode="single")
> msi.prof[[expname]][["convert.ff"]] <- profmem({
+
+         for ( i in seq_len(ncol(iData(msi))) )
+                 msi.ff[,i] <- iData(msi)[,i]
+
+ })
> msi.ff <- vt(msi.ff)
```

```
> ct.mult.ff <- function(A, B, center = ct) {
+         if ( is.vector(A) ) {
+                 A <- t(A)
+                 cbind(ffmatrixmult(A, B)[]) - (sum(A) * ct)
+         } else if ( is.vector(B) ) {
```

```
+                 B <- as.matrix(B)
+                 cbind(ffmatrixmult(A, B)[]) - sum(B * ct)
+         }
+ }
```

```
> msi.prof[[expname]][["ff"]] <- profmem({
+
+         ct <- as.vector(ffapply(X=msi.ff, MARGIN=2, AFUN=mean, RETURN=TRUE)[])
+
+         pca.out <- irlba(msi.ff, nu=0, nv=2, mult=ct.mult.ff)
+
+ })
> file.remove(paste0(backingpath, backingfile))
```

```
> print(msi.prof)

$`3D_Timecourse`
$`3D_Timecourse`$matter
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      175.000       178.100       228.200        50.100       786.791

$`3D_Timecourse`$convert.bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      185.000       189.100       235.800        46.700       432.691

$`3D_Timecourse`$bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      189.900       189.700       330.700       141.000        53.211

$`3D_Timecourse`$convert.ff
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      190.900       191.300       224.100        32.800        76.648

$`3D_Timecourse`$ff
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      192.700       193.500       278.500        85.000      1188.808


$`3D_OSCC`
$`3D_OSCC`$matter
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      275.500       309.500       977.500       668.000      7629.141

$`3D_OSCC`$convert.bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      280.000       465.700      1420.000       954.300      3063.029

$`3D_OSCC`$bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      130.100       142.100       408.500       266.400      9902.989
```

```
$`3DMouseKidney`
$`3DMouseKidney`$matter
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
       346.50        401.80       1476.30       1074.50      12143.69

$`3DMouseKidney`$convert.bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      407.900       713.300      1494.300       781.000      6478.967

$`3DMouseKidney`$bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
       174.60        186.30        617.50        431.20      16163.81


$`3D_Mouse_Pancreas`
$`3D_Mouse_Pancreas`$matter
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      237.100       258.100       628.700       370.600      7966.396

$`3D_Mouse_Pancreas`$convert.bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
      262.600       374.100       630.800       256.700      5490.522

$`3D_Mouse_Pancreas`$bigmemory
   start (MB)   finish (MB) max used (MB) overhead (MB)    time (sec)
       130.10        139.60        303.90        164.30      10330.53
```

5 Session info

- R version 4.1.1 Patched (2021-08-22 r80813), `x86_64-apple-darwin17.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Running under: `macOS Mojave 10.14.6`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: BiocParallel 1.27.17, DBI 1.1.1, Matrix 1.3-4, biglm 0.9-2.1, matter 1.19.1
- Loaded via a namespace (and not attached): BiocGenerics 0.39.2, BiocManager 1.30.16, BiocStyle 2.21.4, ProtGenerics 1.25.1, compiler 4.1.1, digest 0.6.28, evaluate 0.14, fastmap 1.1.0, grid 4.1.1, htmltools 0.5.2, irlba 2.3.3, knitr 1.36, lattice 0.20-45, parallel 4.1.1, rlang 0.4.12, rmarkdown 2.11, tools 4.1.1, xfun 0.27, yaml 2.2.1

References

[1] Janina Oetjen, Kirill Veselkov, Jeramie Watrous, James S McKenzie, Michael Becker, Lena Hauberg-Lotte, Jan Hendrik Kobarg, Nicole Strittmatter, Anna K Mróz, Franziska Hoffmann, Dennis Trede, Andrew Palmer, Stefan Schiffler, Klaus Steinhorst, Michaela Aichler, Robert Goldin, Orlando Guntinas-Lichius, Ferdinand von Eggeling, Herbert Thiele, Kathrin Maedler, Axel Walch, Peter Maass, Pieter C Dorrestein, Zoltan Takats, and Theodore Alexandrov. Benchmark datasets for 3D MALDI- and DESI-imaging mass spectrometry. *GigaScience*, 4(1):2105–8, May 2015.

[2] T. Schramm, A. Hester, I. Klinkert, J. P. Both, R. M. Heeren, A. Brunelle, O. Laprévote, N. Desbenoit, F. Robbe M, M. Stoeckli, B. Spengler, and A. Römpp. imzML – A common data format for the flexible exchange and processing of mass spectrometry imaging data. *Journal of Proteomics*, 75:5106, 2012.