

# *matter*: Supplementary 1 - Simulations and comparative benchmarks

*Kylie A. Bemis*

December 18, 2018

## Contents

1	Introduction . . . . .	1
2	Linear regression . . . . .	2
2.1	Using base R. . . . .	2
2.2	Using <code>bigmemory</code> . . . . .	2
2.3	Using <code>ff</code> . . . . .	3
2.4	Using <code>matter</code> . . . . .	3
3	Principal components analysis . . . . .	4
3.1	Using base R. . . . .	4
3.2	Using <code>bigmemory</code> . . . . .	5
3.3	Using <code>ff</code> . . . . .	5
3.4	Using <code>matter</code> . . . . .	6
4	Summary . . . . .	7
5	Session info . . . . .	7

## 1 Introduction

---

This vignette investigates comparisons in performance between packages *matter* and related packages *bigmemory* [1] and *ff* [2], which also provide infrastructure for working with larger-than-memory datasets in R.

The examples demonstrated here are chosen because both linear regression and principal components analysis are statistical tasks common to many areas of bioinformatics.

The use of simulated data allows us to explore performance in a situation where file format is not an issue.

```
> library(matter)
```

## 2 Linear regression

```
> set.seed(81216)
> chunksize <- 10000
> n <- 1.5e7
> p <- 9
> b <- runif(p)
> names(b) <- paste0("x", 1:p)
> data <- data.frame(y=rnorm(n), check.rows=FALSE)
> for ( nm in names(b) ) {
+   xi <- rnorm(n)
+   data[[nm]] <- xi
+   data[["y"]] <- data[["y"]] + xi * b[nm]
+ }
> fm <- as.formula(paste0("y ~ ", paste0(names(b), collapse=" + ")))
> lm.prof <- list()
```

### 2.1 Using base R

```
> lm.prof[["base"]] <- profmem({
+
+   base.out <- lm(fm, data=data)
+
+ })
> rm(base.out)
> gc()
```

```
> print(lm.prof[["base"]])
```

start (MB)	finish (MB)	max used (MB)	overhead (MB)	time (sec)
1328.700	5582.900	7023.200	1440.300	33.299

### 2.2 Using bigmemory

```
> library(bigmemory)
> library(biganalytics)
> backingfile <- "lm-ex.bin"
> backingpath <- tempdir()
> descriptorfile <- "lm-ex.desc"
> data.bm <- filebacked.big.matrix(nrow=n, ncol=p + 1,
+   backingfile=backingfile,
+   backingpath=backingpath,
+   descriptorfile=descriptorfile,
+   dimnames=list(NULL, c("y", names(b))),
+   type="double")
> for ( nm in names(data) )
```

## **matter: Supplementary 1 - Simulations and comparative benchmarks**

```
+      data.bm[,nm] <- data[[nm]]
> rm(data)
> gc()
```

```
> lm.prof[["bigmemory"]] <- profmem({
+
+      bm.out <- biglm.big.matrix(fm, data=data.bm, chunksize=chunksize)
+
+ })
> rm(bm.out)
> gc()
```

```
> print(lm.prof[["bigmemory"]])

      start (MB)   finish (MB) max used (MB) overhead (MB)   time (sec)
      388.900      389.200      4383.700      3994.500         21.557
```

### 2.3 Using `ff`

```
> library(ff)
> library(ffbase)
> data.ff <- ff(filename=paste0(backingpath, "/", backingfile),
+      vmode="double", dim=c(n, p + 1),
+      dimnames=list(NULL, c("y", names(b))))
> data.ff <- as.ffdf(data.ff)
```

```
> lm.prof[["ff"]] <- profmem({
+
+      ff.out <- bigglm(fm, data=data.ff, chunksize=chunksize)
+
+ })
> rm(ff.out)
> gc()
```

```
> print(lm.prof[["ff"]])

      start (MB)   finish (MB) max used (MB) overhead (MB)   time (sec)
      392.300      393.300      1986.800      1593.500         56.987
```

### 2.4 Using `matter`

```
> data.m <- matter(paths=paste0(backingpath, "/", backingfile),
+      datamode="double", nrow=n, ncol=p + 1,
+      dimnames=list(NULL, c("y", names(b))))
```

```
> lm.prof[["matter"]] <- profmem({
+
+ 
```

## ***matter*: Supplementary 1 - Simulations and comparative benchmarks**

```
+      m.out <- bigglm(fm, data=data.m, chunksize=chunksize)
+
+ })
> rm(m.out)
> gc()
```

```
> print(lm.prof[["matter"]])
```

start (MB)	finish (MB)	max used (MB)	overhead (MB)	time (sec)
393.400	393.600	1054.100	660.500	47.175

## 3 Principal components analysis

---

```
> library(irlba)
> set.seed(81216)
> n <- 1.5e6
> p <- 100
> data <- matrix(nrow=n, ncol=p)
> for ( i in 1:10 )
+   data[,i] <- (1:n)/n + rnorm(n)
> for ( i in 11:20 )
+   data[,i] <- (n:1)/n + rnorm(n)
> for ( i in 21:p )
+   data[,i] <- rnorm(n)
> pca.prof <- list()
```

This again creates a 1.2 GB dataset in memory.

### 3.1 Using base R

First, we demonstrate

```
> pca.prof[["base"]] <- profmem({
+
+   base.out <- svd(data, nu=0, nv=2)
+
+ })
> rm(base.out)
> gc()
```

```
> print(pca.prof[["base"]])
```

start (MB)	finish (MB)	max used (MB)	overhead (MB)	time (sec)
1593.600	1593.700	3994.100	2400.400	66.231

## 3.2 Using bigmemory

```
> library(bigalgebra)
> backingfile <- "pca-ex.bin"
> backingpath <- tempdir()
> descriptorfile <- "pca-ex.desc"
> data.bm <- filebacked.big.matrix(nrow=n, ncol=p,
+   backingfile=backingfile,
+   backingpath=backingpath,
+   descriptorfile=descriptorfile,
+   type="double")
> for ( i in seq_len(ncol(data)) )
+   data.bm[,i] <- data[,i]
> rm(data)
> gc()
> mult.bm <- function(A, B) {
+   if ( is.vector(A) )
+     A <- t(A)
+   if ( is.vector(B) )
+     B <- as.matrix(B)
+   cbind((A %*% B)[,])
+ }
```

```
> pca.prof[["bigmemory"]] <- profmem({
+
+   bm.out <- irlba(data.bm, nu=0, nv=2, mult=mult.bm)
+
+ })
> rm(bm.out)
> gc()
```

```
> print(pca.prof[["bigmemory"]])
```

start (MB)	finish (MB)	max used (MB)	overhead (MB)	time (sec)
393.900	406.400	3110.000	2703.600	15.391

## 3.3 Using ff

```
> library(bootSVD)
> data.ff <- ff(filename=paste0(backingpath, "/", backingfile),
+   vmode="double", dim=c(n, p))
> mult.ff <- function(A, B) {
+   if ( is.vector(A) )
+     A <- t(A)
+   if ( is.vector(B) )
+     B <- as.matrix(B)
+   cbind(ffmatrixmult(A, B)[,])
+ }
```

## ***matter*: Supplementary 1 - Simulations and comparative benchmarks**

Linear regression				Principle component analysis			
Method	Mem. used	Mem. overhead	Time	Method	Mem. used	Mem. overhead	
R matrices + lm	7 GB	1.4 GB	33 sec	R matrices + svd	3.9 GB	2.4 GB	6
bigmemory + biglm	4.4 GB	3.9 GB	21 sec	bigmemory + irlba	3.1 GB	2.7 GB	1
ff + biglm	1.9 GB	1.6 GB	57 sec	ff + irlba	1.8 GB	1.4 GB	13
matter + biglm	1 GB	660 MB	47 sec	matter + irlba	890 MB	490 MB	11

**Table 1: Comparative performance of *matter* for linear regression and calculation of the first two principal components on simulated datasets of 1.2 GB**

Memory overhead is the maximum memory used during the execution minus the memory in use upon completion.

```
> pca.prof[["ff"]] <- profmem({
+
+   ff.out <- irlba(data.ff, nu=0, nv=2, mult=mult.ff)
+
+ })
> rm(ff.out)
> gc()
```

```
> print(pca.prof[["ff"]])

start (MB)  finish (MB) max used (MB) overhead (MB)  time (sec)
    394.500    407.200    1790.400    1383.200      130.828
```

### 3.4 Using *matter*

```
> library(matter)
> data.m <- matter(paths=paste0(backingpath, "/", backingfile),
+   datamode="double", nrow=n, ncol=p)
```

```
> pca.prof[["matter"]] <- profmem({
+
+   m.out <- irlba(data.m, nu=0, nv=2, fastpath=FALSE)
+
+ })
> rm(m.out)
> gc()
```

```
> print(pca.prof[["matter"]])

start (MB)  finish (MB) max used (MB) overhead (MB)  time (sec)
    395.400    407.400    893.900    486.500      110.884
```

## 4 Summary

---

Table 1 demonstrates that *matter* typically uses less memory than both *bigmemory* and *ff*. Additionally, it outperforms *ff* in speed. The reason for *bigmemory*'s superior speed is likely its use of `mmap` to map the on-disk data to virtual memory. This allows it to perform faster on datasets that can fit into available memory. However, this also uses more memory, because the much of the data ends up being loaded into memory. A comparison on real datasets that are much larger than memory (in the vignette “Supplementary 2 - 3D mass spectrometry imaging case study”) demonstrate the *matter* can be faster than *bigmemory* on datasets too large to be fully loaded into memory.

## 5 Session info

---

- R version 3.5.1 Patched (2018-07-24 r75008), x86\_64-w64-mingw32
- Locale: LC\_COLLATE=C, LC\_CTYPE=English\_United States.1252, LC\_MONETARY=English\_United States.1252, LC\_NUMERIC=C, LC\_TIME=English\_United States.1252
- Running under: Windows Server 2012 R2 x64 (build 9600)
- Matrix products: default
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: DBI 1.0.0, biglm 0.9-1, matter 1.8.3
- Loaded via a namespace (and not attached): BiocGenerics 0.28.0, BiocManager 1.30.4, BiocStyle 2.10.0, Matrix 1.2-15, Rcpp 1.0.0, compiler 3.5.1, digest 0.6.18, evaluate 0.12, grid 3.5.1, htmltools 0.3.6, irlba 2.3.2, knitr 1.21, lattice 0.20-38, parallel 3.5.1, rmarkdown 1.11, tools 3.5.1, xfun 0.4, yaml 2.2.0

## References

- [1] Michael J. Kane, John Emerson, and Stephen Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. URL: <http://www.jstatsoft.org/v55/i14/>.
- [2] Daniel Adler, Christian Gladser, Oleg Nenadic, Jens Oehlschladgel, and Walter Zucchini. *ff: memory-efficient storage of large data on disk and fast access functions*, 2014. R package version 2.2-13. URL: <https://CRAN.R-project.org/package=ff>.