

# DelayedMatrixStats

Porting the matrixStats API to work with  
DelayedMatrix objects

Peter Hickey (@PeteHaitch)

2017-07-26

---

# Why matrixStats?

**matrixStats** by Henrik Bengtsson and co. on CRAN since 2009

# Why matrixStats?

**matrixStats** by Henrik Bengtsson and co. on CRAN since 2009

Lots of useful col/row summary functions

```
grep("^col", getNamespaceExports("matrixStats"), value = TRUE)
#> [1] "colMadDiffs"      "colCummins"      "colRanks"
#> [4] "colWeightedVars" "colQuantiles"    "colDiffs"
#> [7] "colCumprods"     "colSds"          "colCollapse"
#> [10] "colVars"         "colAnyMissings" "colWeightedSds"
#> [13] "colCummaxs"     "colAlls"         "colVarDiffs"
#> [16] "colIQRs"        "colMins"         "colWeightedMedians"
#> [19] "colLogSumExps"  "colAvgPerRowSet" "colSdDiffs"
#> [22] "colIQRDiffs"   "colSums2"        "colCumsums"
#> [25] "colTabulates"   "colMedians"      "colOrderStats"
#> [28] "colWeightedMads" "colMaxs"         "colCounts"
#> [31] "colWeightedMeans" "colMeans2"       "colProds"
#> [34] "colRanges"     "colAnyNAs"       "colAnys"
#> [37] "colMads"
```

# Optimised row/column operations on *matrix* objects

```
# Simulate some zero-inflated count data
matrix <- matrix(sample(0:100, 20000 * 10000, replace = TRUE),
                 nrow = 20000,
                 ncol = 10000)
matrix[sample(length(matrix), length(matrix) * 0.6)] <- 0L

library(matrixStats)
benchmark(apply(matrix, 2, median),
          colMedians(matrix),
          times = 10)

#>                               expr Median time (s) Mem alloc (MB)
#>  apply(matrix, 2, median)                8.22           4802.2
#>  colMedians(matrix)                      1.99              0.3
```

# Why matrixStats?

## Optimised row/column operations on *matrix* objects

```
j <- c(2001:3000, 5001:5500)
benchmark(colSums(matrix[, j]),
          colSums2(matrix, cols = j),
          times = 10)
#>           expr Median time (ms) Mem alloc (MB)
#>   colSums(matrix[, j])          759.0         120.1
#> colSums2(matrix, cols = j)         53.3           0.2
```

# Big data blues

- You've got matrix-like data but too large for in-memory *matrix* :(

# Big data blues

- You've got matrix-like data but too large for in-memory *matrix* :(

## *DelayedMatrix!*

- A wrapper around a matrix-like object
- Data can be in memory or on disk
- *DelayedMatrix* works as an assay in a *SummarizedExperiment*
- *DelayedMatrix* supports the standard & familiar *matrix* API\*
  - `[`
  - `dim()`
  - `dimnames()`
  - `t()`
  - `log()`
  - **`colSums()`**
  - ...

[\*] But not subassignment

# *DelayedMatrix* backends

## In-memory backends

```
DelayedMatrix <- DelayedArray::DelayedArray(matrix)
pryr::object_size(DelayedMatrix)
#> 800 MB
```

```
DelayeddgCMatrix <- DelayedArray(as(matrix, "dgCMatrix"))
pryr::object_size(DelayeddgCMatrix) # Larger than dense version!
#> 951 MB
```

```
RleMatrix <- RleArray(Rle(matrix), dim = dim(matrix))
pryr::object_size(RleMatrix) # Low RLE compressibility
#> 1.01 GB
```

```
TricksyRleMatrix <- as(matrix, "RleMatrix") # Uses tricky tricks
pryr::object_size(TricksyRleMatrix) # Tricky tricks in play
#> 634 MB
```



# *DelayedMatrix* backends

## On-disk backends

```
HDF5Matrix <- HDF5Array::writeHDF5Array(matrix)
pryr::object_size(HDF5Matrix)
#> 2.39 kB
file_size(HDF5Matrix@seed@file)
#> 165 MB

matterMatrix <- matterArray::writeMatterArray(matrix)
pryr::object_size(matterMatrix)
#> 9.63 kB
file_size(matterMatrix@seed@matter@paths)
#> 800 MB
```

# Why DelayedMatrixStats?

# Why DelayedMatrixStats?



---

# Why DelayedMatrixStats?

- Support **matrixStats** API for *DelayedMatrix* and derived classes
- Reduce friction between using *matrix* or *DelayedMatrix*

# Why DelayedMatrixStats?

- Support **matrixStats** API for *DelayedMatrix* and derived classes
- Reduce friction between using *matrix* or *DelayedMatrix*

## Initial release aim

General 'block-processing' method to work for *DelayedMatrix* and arbitrary derived classes

# Why DelayedMatrixStats?

- Support **matrixStats** API for *DelayedMatrix* and derived classes
- Reduce friction between using *matrix* or *DelayedMatrix*

## Initial release aim

General 'block-processing' method to work for *DelayedMatrix* and arbitrary derived classes

## Subsequent releases

'Backend-aware' optimised methods

# Why DelayedMatrixStats?

Yay, same syntax works regardless of backend!

```
benchmark(colMedians(matrix),  
          colMedians(DelayedMatrix),  
          colMedians(DelayeddgCMatrix),  
          colMedians(RleMatrix),  
          colMedians(TricksyRleMatrix),  
          colMedians(HDF5Matrix),  
          colMedians(matterMatrix),  
          times = 10)
```

```
#>           expr Median time (s) Mem alloc (MB)  
#>           colMedians(matrix)           1.99           0.3  
#>           colMedians(DelayedMatrix)       1.94           0.3  
#> colMedians(DelayeddgCMatrix)           16.70          10402.7  
#>           colMedians(RleMatrix)           24.10           7295.1  
#> colMedians(TricksyRleMatrix)           66.00          34284.8  
#>           colMedians(HDF5Matrix)          22.00           5396.6  
#>           colMedians(matterMatrix)         7.15           4052.1
```

```
# Aside: apply(DelayedMatrix, 2, median) currently doesn't work
```

# Why DelayedMatrixStats?

## Backend-aware methods can improve performance

```
CS <- function(x, j) colSums(x[, j]) # DelayedArray
CS2 <- function(x, j) colSums2(x, cols = j) # DelayedMatrixStats
j <- c(2001:3000, 5001:5500)
benchmark(CS(DelayedMatrix, j), # Block-processing
          CS2(DelayedMatrix, j), # Backend-aware
          CS(DelayedddgCMatrix, j), # Block-processing
          CS2(DelayedddgCMatrix, j), # Backend-aware
          CS(RleMatrix, j), # Block-processing
          CS2(RleMatrix, j), # Backend-aware
          times = 10)

#>          expr Median time (ms) Mem alloc (MB)
#> CS(DelayedMatrix, j)          694.0          482.7
#> CS2(DelayedMatrix, j)          52.7           0.2
#> CS(DelayedddgCMatrix, j)      6520.0         1103.3
#> CS2(DelayedddgCMatrix, j)     312.0          142.6
#> CS(RleMatrix, j)             2770.0         1087.0
#> CS2(RleMatrix, j)            234.0           0.1
```



# For more

**DelayedMatrixStats:** <https://github.com/PeteHaitch/DelayedMatrixStats>

**matter:** Developed by Kylie A. Bemis  
<https://bioconductor.org/packages/matter/>

**matterArray:** <https://github.com/PeteHaitch/matterArray>

**Slides:** <http://peterhickey.org/presentations/>

**GitHub & Twitter:** [@PeteHaitch](#)

