DelayedMatrixStats

Porting the matrixStats API to work with DelayedMatrix objects

Peter Hickey

2017-07-26

Why **matrixStats**?

matrixStats by Henrik Bengtsson and co. on CRAN since 2009

> "Functions that Apply to Rows and Columns of Matrices"

Optimised row/column operations on *matrix* objects

```r
# Simulate some zero-inflated count data
matrix <- matrix(sample(0:100, 20000 * 100, replace = TRUE),
                 nrow = 200,
                 ncol = 10000)
matrix[sample(length(matrix), length(matrix) * 0.6)] <- 0L

# matrixStats has fast, memory-efficient column summaries
benchmark(apply(matrix, 2, median),
          matrixStats::colMedians(matrix),
          times = 1)
#>                              expr Median time (ms) Mem alloc (MB)
#>          apply(matrix, 2, median)            774.0        50.2000
#>   matrixStats::colMedians(matrix)             28.7         0.0825
```

Why **matrixStats**?

Optimised row/column operations on *matrix* objects

```r
library(matrixStats)

# matrixStats optimised for subsetted calculations
j <- c(2001:3000, 5001:5500)
benchmark(colSums(matrix[, j]),
          colSums2(matrix, cols = j),
          times = 1)
#>                           expr Median time (ms) Mem alloc (MB)
#>         colSums(matrix[, j])             5.22         1.2100
#>   colSums2(matrix, cols = j)             1.19         0.0171
```

Why **matrixStats**?

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"     "colAvgsPerRowSet"  "colSdDiffs"
#> [22] "colIQRDiffs"       "colSums2"          "colCumsums"
#> [25] "colTabulates"      "colMedians"        "colOrderStats"
#> [28] "colWeightedMads"   "colMaxs"           "colCounts"
#> [31] "colWeightedMeans"  "colMeans2"         "colProds"
#> [34] "colRanges"         "colAnyNAs"         "colAnys"
#> [37] "colMads"
```

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)
#> 8 MB

DelayeddgCMatrix <- DelayedArray(as(matrix, "dgCMatrix"))
pryr::object_size(DelayeddgCMatrix) # Larger than dense version!
#> 9.55 MB

RleMatrix <- RleArray(Rle(matrix), dim = dim(matrix))
pryr::object_size(RleMatrix)        # Low RLE compressibility
#> 10.1 MB

TricksyRleMatrix <- as(matrix, "RleMatrix") # Uses tricksy tricks
pryr::object_size(TricksyRleMatrix) # Tricksy tricks in play
#> 6.34 MB
```

DelayedMatrix backends

On-disk backends

```
HDF5Matrix <- HDF5Array::writeHDF5Array(matrix)
pryr::object_size(HDF5Matrix)
#> 2.39 kB
file_size(HDF5Matrix@seed@file)
#> 1.63 MB

matterMatrix <- matterArray::writeMatterArray(matrix)
pryr::object_size(matterMatrix)
#> 9.63 kB
file_size(matterMatrix@seed@matter@paths)
#> 8 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 = 1)
#>                          expr Median time (ms) Mem alloc (MB)
#>            colMedians(matrix)             26.0         0.0860
#>     colMedians(DelayedMatrix)             25.2         0.0932
#>  colMedians(DelayeddgCMatrix)            850.0       107.0000
#>         colMedians(RleMatrix)            287.0        72.4000
#>  colMedians(TricksyRleMatrix)           1550.0       333.0000
#>        colMedians(HDF5Matrix)            286.0        49.9000
#>      colMedians(matterMatrix)            139.0        40.5000

# 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(DelayeddgCMatrix, j),   # Block-processing
          CS2(DelayeddgCMatrix, j),  # Backend-aware
          CS(RleMatrix, j),          # Block-processing
          CS2(RleMatrix, j),         # Backend-aware
          times = 1)
#>                        expr Median time (ms) Mem alloc (MB)
#>       CS(DelayedMatrix, j)            20.40         4.9200
#>      CS2(DelayedMatrix, j)             2.55         0.0244
#>    CS(DelayeddgCMatrix, j)           138.00        11.2000
#>   CS2(DelayeddgCMatrix, j)            20.30         1.4800
#>           CS(RleMatrix, j)            33.30        10.9000
#>          CS2(RleMatrix, j)            25.90         0.6650
```

For more

DelayedMatrixStats: https://github.com/PeteHaitch/DelayedMatrixStats

matter developed by Kylie A. Bemis and available on Bioconductor

matterArray: https://github.com/PeteHaitch/matterArray

Slides: http://peterhickey.org/presentations/

GitHub & Twitter: @PeteHaitch