Background

It turns out that there is some documentation on `dgCMatrix` objects within the `Matrix` package. One can access it using the following code:

```
library(Matrix)
?`dgCMatrix-class`
```

According to the documentation, the `dgCMatrix` class

> …is a class of sparse numeric matrices in the compressed, sparse, column-oriented format. In this implementation the non-zero elements in the columns are sorted into increasing row order. `dgCMatrix` is the "standard" class for sparse numeric matrices in the `Matrix` package.

An example

We'll use a small matrix as a running example in this post:

```
library(Matrix)
M <- Matrix(c(0, 0,  0, 2,
              6, 0, -1, 5,
              0, 4,  3, 0,
              0, 0,  5, 0),
            byrow = TRUE, nrow = 4, sparse = TRUE)
rownames(M) <- paste0("r", 1:4)
colnames(M) <- paste0("c", 1:4)
M
# 4 x 4 sparse Matrix of class "dgCMatrix"
#    c1 c2 c3 c4
# r1  .  .  .  2
# r2  6  . -1  5
# r3  .  4  3  .
# r4  .  .  5  .
```

Running `str` on `x` tells us that the `dgCMatrix` object has 6 slots. (To learn more about slots and S4 objects, see this section from Hadley Wickham's *Advanced R*.)

```
str(M)
# Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
# ..@ i       : int [1:7] 1 2 1 2 3 0 1
# ..@ p       : int [1:5] 0 1 2 5 7
# ..@ Dim     : int [1:2] 4 4
# ..@ Dimnames:List of 2
# .. ..$ : chr [1:4] "r1" "r2" "r3" "r4"
# .. ..$ : chr [1:4] "c1" "c2" "c3" "c4"
# ..@ x       : num [1:7] 6 4 -1 3 5 2 5
# ..@ factors : list()
```

`x`, `i` and `p`

If a matrix `M` has `nn` non-zero entries, then its `x` slot is a vector of length `nn` containing all the non-zero values in the matrix. The non-zero elements in column 1 are listed first (starting from the top and ending at the bottom), followed by column 2, 3 and so on.

```
M
# 4 x 4 sparse Matrix of class "dgCMatrix"
#    c1 c2 c3 c4
# r1  .  .  .  2
# r2  6  . -1  5
# r3  .  4  3  .
# r4  .  .  5  .

M@x
# [1]  6  4 -1  3  5  2  5

as.numeric(M)[as.numeric(M) != 0]
```

```
# [1]  6  4 -1  3  5  2  5
```

The `i` slot is a vector of length `nn`. The `k`th element of `M@i` is the row index of the `k`th non-zero element (as listed in `M@x`). ***One big thing to note here is that the first row has index ZERO, unlike R's indexing convention.*** In our example, the first non-zero entry, 6, is in the second row, i.e. row index 1, so the first entry of `M@i` is 1.

```
M
# 4 x 4 sparse Matrix of class "dgCMatrix"
#    c1 c2 c3 c4
# r1  .  .  .  2
# r2  6  . -1  5
# r3  .  4  3  .
# r4  .  .  5  .

M@i
# [1] 1 2 1 2 3 0 1
```

If the matrix has `nvars` columns, then the `p` slot is a vector of length `nvars + 1`. ***If we index the columns such that the first column has index ZERO,*** then `M@p[1] = 0`, and `M@p[j+2] - M@p[j+1]` gives us the number of non-zero elements in column `j`.

In our example, when `j = 2`, `M@p[2+2] - M@p[2+1] = 5 - 2 = 3`, so there are 3 non-zero elements in column index 2 (i.e. the third column).

```
M
# 4 x 4 sparse Matrix of class "dgCMatrix"
#    c1 c2 c3 c4
# r1  .  .  .  2
# r2  6  . -1  5
# r3  .  4  3  .
# r4  .  .  5  .

M@p
# [1] 0 1 2 5 7
```

With the `x`, `i` and `p` slots, one can reconstruct the entries of the matrix.

`Dim` and `Dimnames`

These two slots are fairly obvious. `Dim` is a vector of length 2, with the first and second entries denoting the number of rows and columns the matrix has respectively. `Dimnames` is a list of length 2: the first element being a vector of row names (if present) and the second being a vector of column names (if present).

`factors`

This slot is probably the most unusual of the lot, and its documentation was a bit difficult to track down. From the CRAN documentation, it looks like `factors` is

> … [an] Object of class "list" – a list of factorizations of the matrix. Note that this is typically empty, i.e., `list()`, initially and is updated ***automagically*** whenever a matrix factorization is computed.

My understanding is if we perform any matrix factorizations or decompositions on a `dgCMatrix` object, it stores the factorization under `factors` so that if asked for the factorization again, it can return the cached value instead of recomputing the factorization. Here is an example:

```
M@factors
# list()

Mlu <- lu(M)  # perform triangular decomposition
str(M@factors)
# List of 1
# $ LU:Formal class 'sparseLU' [package "Matrix"] with 5 slots
# .. ..@ L  :Formal class 'dtCMatrix' [package "Matrix"] with 7 slots
# .. .. .. ..@ i       : int [1:4] 0 1 2 3
# .. .. .. ..@ p       : int [1:5] 0 1 2 3 4
# .. .. .. ..@ Dim     : int [1:2] 4 4
```

```
# .. .. .. ..@ Dimnames:List of 2
# .. .. .. .. ..$ : chr [1:4] "r2" "r3" "r4" "r1"
# .. .. .. .. ..$ : NULL
# .. .. .. ..@ x        : num [1:4] 1 1 1 1
# .. .. .. ..@ uplo     : chr "U"
# .. .. .. ..@ diag     : chr "N"
# .. ..@ U  :Formal class 'dtCMatrix' [package "Matrix"] with 7 slots
# .. .. .. ..@ i        : int [1:7] 0 1 0 1 2 0 3
# .. .. .. ..@ p        : int [1:5] 0 1 2 5 7
# .. .. .. ..@ Dim      : int [1:2] 4 4
# .. .. .. ..@ Dimnames:List of 2
# .. .. .. .. ..$ : NULL
# .. .. .. .. ..$ : chr [1:4] "c1" "c2" "c3" "c4"
# .. .. .. ..@ x        : num [1:7] 6 4 -1 3 5 5 2
# .. .. .. ..@ uplo     : chr "U"
# .. .. .. ..@ diag     : chr "N"
# .. ..@ p  : int [1:4] 1 2 3 0
# .. ..@ q  : int [1:4] 0 1 2 3
# .. ..@ Dim: int [1:2] 4 4
```

Here is an example which shows that the decomposition is only performed once:

```
set.seed(1)
M <- runif(9e6)
M[sample.int(9e6, size = 8e6)] <- 0
M <- Matrix(M, nrow = 3e3, sparse = TRUE)

system.time(lu(M))
#   user  system elapsed
# 13.527   0.161  13.701

system.time(lu(M))
#   user  system elapsed
#      0       0       0
```
