Introduction to DatABEL

Yurii S. Aulchenko, Stepan Yakovenko

February 10, 2011

Contents

1 Introduction

This vignette demonstrates the use of all major DatABEL functions. Central to DatABEL library is the databel class, which is defined as following:

```
setClass(
    Class = "databel",
    representation = representation(
        usedRowIndex = "integer",
        usedColIndex = "integer",
        uninames = "list",
        backingfilename = "character",
        cachesizeMb = "integer",
        data = "externalptr"
    ),
    package = "DatABEL"
);
```

here, data is external pointer to an instance of FilteredMatrix class of filevector library, usedRowIndex and usedColIndex keep the indexes of not masked columns

and rows, `backingfilename` is the base name of the `filevector` data/index files, and `cachesizeMb` specifies the amount of RAM used for cache. The `uninames` list specifies whether the column and/or row names are unique and thus may be used to access the data.

The methods defined for `databel` class are similar to that defined for standard matrices and allow to (throughout, `DAdata` refers to an object of `databel` class):

- Obtain information about underlying data (`show`, `dim`, `dimnames`, `get_dimnames`, `length`, `backingfilename` and `cachesizeMb`). The function `get_dimnames` returns a list with row and column names defined for the data object; the function `dimnames` does so if the names are unique; in case row/column names are not unique `NULL` is returned for that dimension.
- Set some attributes (`dimnames<-`, `set_dimnames<-`, `cachesizeMb<-` and `setReadOnly<-`).
- Connect and disconnect R object of `databel`-class to/from the underlying binary data (`connect` and `disconnect`; these functions destroy or initiate an instance of `FilteredMatrix`.
- Save (sub-set) of `databel` matrix as new binary set of files (`save_as`) or export to plain text files (`databel2text`)
- Obtain sub-sets of a `databel` object (operation `[`).
- Replace values in the matrix (operation `[<-`)
- Coercion of `databel` matrix to standard R matrix and vector and coercion of R matrix to `databel` matrix.

Internally, `databel` data may comprise eight different types (float, double, signed/unsigned (short) int, signed/unsigned byte). In C++, two of these (double and float) have support for missing values ('not a number'). For the rest, we reserved the maximal value to texttt for the missing data.

Additionally functions to convert plain text files to `databel` format (`text2databel`) and to export `databel` data to plain text (`databel2text`) are provided. Another function (`apply2dfo`) is similar to standard R `apply` and allows application of user-defined function to all rows/columns of the data.

2 Conversion of the data to `databel` format, initialization of `databel` objects, and value modifications

To start using `DatABEL` you first need to load the library:

```
> library(DatABEL)
```

```
DatABEL v. 0.9-3 (January 28, 2011) loaded

Installed DatABEL version (0.9-3) is not the same as stable
version available from CRAN (0.9-2). Unless used intentionally,
consider updating to the latest CRAN version. For that, use
'install.packages("DatABEL")', or ask your system administrator
to update the package.

YOU APPEAR TO WORK ON 32-BIT SYSTEM. LARGE FILES ARE NOT SUPPORTED.
```

We will first create an R matrix and will convert that to `databel` format. For that, create R matrix:

```
> matr <- matrix(c(1:12), ncol = 3, nrow = 4)
> matr[3, 2] <- NA
> matr

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3   NA   11
[4,]    4    8   12

> dimnames(matr) <- list(paste("row", 1:4, sep = ""), paste("col",
+     1:3, sep = ""))
> matr

     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3   NA   11
row4    4    8   12
```

Conversion from R matrix to `databel` may be performed in two ways, using generic 'as' function or `matrix2databel` function. The difference is that when using 'as' the backing data file is named by generating a random name and the type used for storage is 'double', while with `matrix2databel` function the user may choose the backing data file name and the type of the data him or herself. Thus, 'as' should be used to create temporary `databel` objects:

```
> list.files(pattern = "*.fv?")

character(0)

> dat1 <- as(matr, "databel")

coersion from 'matrix' to 'databel' of type DOUBLE ; object connected to file ./tmp57633

> list.files(pattern = "*.fv?")
```

```
[1] "tmp57633.fvd" "tmp57633.fvi"
```

You can see that after application of `as` method, two files containing data backing the 'dat1' have appeared.

The 'show' method shows basic information for the object:

```
> dat1

uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = ./tmp57633
cachesizeMb = 64
number of columns (variables) =  3
number of rows (observations) =  4
usedRowIndex: 1  2  3  4
usedColIndex: 1  2  3
Upper-left 3 columns and  4 rows:
     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3  NaN   11
row4    4    8   12
```

Note that for big matrices only summaries and small part of the data will appear on the screen.

To keep the naming of the backing files, underlying data type and other details under control, use `matrix2databel` function:

```
> dat2 <- matrix2databel(matr, filename = "matr", cachesizeMb = 16,
+     type = "UNSIGNED_CHAR", readonly = FALSE)
> dat2

uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = matr
cachesizeMb = 16
number of columns (variables) =  3
number of rows (observations) =  4
usedRowIndex: 1  2  3  4
usedColIndex: 1  2  3
Upper-left 3 columns and  4 rows:
     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3  NaN   11
row4    4    8   12
```

You can see that now the backing files are `matr.fvd` and `matr.fvi`:

```
> list.files(pattern = "*.fv?")

[1] "matr.fvd"     "matr.fvi"     "tmp57633.fvd" "tmp57633.fvi"
```

If you try to create a new object with the same backing files, an error will appear.

A new `databel` object can be initialized directly from the backing file:

```
> dat3 <- databel("matr")
> dat3

uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = matr
cachesizeMb = 64
number of columns (variables) =  3
number of rows (observations) =  4
usedRowIndex: 1  2  3  4
usedColIndex: 1  2  3
Upper-left 3 columns and  4 rows:
     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3  NaN   11
row4    4    8   12
```

A `databel` object can also be created from a text file. First, we will create a text file

```
> write.table(matr, "matr.txt", row.names = TRUE, col.names = TRUE,
+     quote = FALSE)
```

and then convert that to `databel` format

```
> dat4 <- text2databel("matr.txt", outfile = "matr1", R_matrix = TRUE,
+     type = "UNSIGNED_INT")

Options in effect:
         --infile    = matr.txt
         --outfile   = matr1
         --skiprows  = 1
         --skipcols  = 1
         --cnrow     = ON, using line 1 of 'matr.txt'
         --rncol     = ON, using column 1 of 'matr.txt'
         --transpose = OFF
         --Rmatrix   = ON
```

```
         --nanString = NA
Number of lines in source file is 5
Number of words in source file is 3
skiprows = 1
cnrow = 1
skipcols = 1
rncol = 1
Rmatrix = 1
numWords = 3
Creating file with numRows = 4
Creating file with numColumns = 3
Transposing matr1_fvtmp => matr1.
text2fvf finished.
```

```
> dat4
```

```
uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = matr1
cachesizeMb = 64
number of columns (variables) =  3
number of rows (observations) =  4
usedRowIndex: 1  2  3  4
usedColIndex: 1  2  3
Upper-left 3 columns and  4 rows:
     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3  NaN   11
row4    4    8   12
```

Finally, a `databel` object can be initialized from other `databel` object

```
> dat5 <- dat4
```

or, through use of `'['`

```
> dat6 <- dat1[c("row1", "row3"), c("col1", "col2")]
> dat6
```

```
uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = ./tmp57633
cachesizeMb = 64
number of columns (variables) =  2
number of rows (observations) =  2
```

```
usedRowIndex: 1  3
usedColIndex: 1  2
Upper-left 2 columns and  2 rows:
     col1 col2
row1    1    5
row3    3  NaN
```

Thus, at the moment we have generated five `databel` objects containing identical data (though underlying type is different: double, unsigned byte and unsigned int) and one object ('dat6') which contains subset of the data. Objects 'dat1' and 'dat6' are using the same backing data file `./tmp57633`, objects 'dat4' and 'dat5' are connected to `matr1`, and 'dat2' and 'dat3' are connected to `matr`.

The data contained in `databel` matrices may be modified by use of `[<-` method:

```
> dat1[1, 1] <- 321
```

Note that because 'dat1' and 'dat6' are connected to the same binary data, modification of 'dat1' leads automatically to modification of 'dat6':

```
> dat6
```

```
uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = ./tmp57633
cachesizeMb = 64
number of columns (variables) =  2
number of rows (observations) =  2
usedRowIndex: 1  3
usedColIndex: 1  2
Upper-left 2 columns and  2 rows:
     col1 col2
row1  321    5
row3    3  NaN
```

To avoid read/write conflicts, all consecutive objects based on the same backing files will be connected in read-only mode (so that trying '`dat6[1,1] <- 123`' will generate an error). We will show how to work around this situation at the end of the next section.

3 Obtain and modifying attributes

Several standard methods defined for matrix are defined for `databel` matrices as well. For example

```
> dim(dat1)
```

```
[1] 4 3

> length(dat1)

[1] 12

> dimnames(dat1)

[[1]]
[1] "row1" "row2" "row3" "row4"

[[2]]
[1] "col1" "col2" "col3"

> colnames(dat1)

[1] "col1" "col2" "col3"

> rownames(dat1)

[1] "row1" "row2" "row3" "row4"
```

The method `dimnames<-` may be used to modify the names:

```
> dimnames(dat1) <- list(paste("ID", 1:4, sep = ""), paste("SNP",
+     1:3, sep = ""))
> dimnames(dat1)

[[1]]
[1] "ID1" "ID2" "ID3" "ID4"

[[2]]
[1] "SNP1" "SNP2" "SNP3"
```

Additional methods defined for `databel` matrices allow to obtain information about backing file name

```
> backingfilename(dat1)

[1] "./tmp57633"
```

and size of cache used

```
> cachesizeMb(dat1)

[1] 64
```

The size of cache can be modified by

```
> cachesizeMb(dat1) <- 1
> cachesizeMb(dat1)
```

```
[1] 1
```

A method `get_dimnames` is defined to obtain row/column names in case these are not uniqie. To demonstrate use of this method, we need first to create a `databel` matrix with non-unique dimnames. To set such not unique names, we will use method `set_dimnames`:

```
> set_dimnames(dat1) <- list(dimnames(dat1)[[1]], c("duplicate",
+     "col2", "duplicate"))
```

Now `dimnames` returns `NULL` for the second dimension names:

```
> dimnames(dat1)
```

```
[[1]]
[1] "ID1" "ID2" "ID3" "ID4"

[[2]]
NULL
```

while `get_dimnames` still allows access to the names:

```
> get_dimnames(dat1)
```

```
[[1]]
[1] "ID1" "ID2" "ID3" "ID4"

[[2]]
[1] "duplicate" "col2"      "duplicate"
```

Finally, the read-only flag can be modifed. The following code demonstrates how to modify the 'dat6' object:

```
> disconnect(dat1)
> setReadOnly(dat6) <- FALSE
> dat6[1, 1] <- 123
> dat6
```

```
uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = ./tmp57633
cachesizeMb = 64
number of columns (variables) =  2
number of rows (observations) =  2
usedRowIndex: 1  3
usedColIndex: 1  2
Upper-left 2 columns and  2 rows:
    duplicate col2
ID1       123    5
ID3         3  NaN
```

```
> dat1

uninames$unique.names = FALSE
uninames$unique.rownames = TRUE
uninames$unique.colnames = FALSE
backingfilename = ./tmp57633
cachesizeMb = 1
number of columns (variables) =  3
number of rows (observations) =  4
usedRowIndex: 1  2  3  4
usedColIndex: 1  2  3
Upper-left 3 columns and  4 rows:
    [,1] [,2] [,3]
ID1  123    5    9
ID2    2    6   10
ID3    3  NaN   11
ID4    4    8   12
```

4 Coersion and exports

A standard R matrix can be obtained from a `databel` matrix by use of function 'as':

```
> newm <- as(dat2, "matrix")
> class(newm)

[1] "matrix"

> class(newm[1, 1])

[1] "numeric"

> newm

     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3  NaN   11
row4    4    8   12
```

A data from `databel` matrix may be exported to a text file using function

```
> databel2text(dat2, file = "dat2.txt")

uninames$unique.names = TRUE
uninames$unique.rownames = TRUE
uninames$unique.colnames = TRUE
backingfilename = matr
```

```
cachesizeMb = 16
number of columns (variables) =  3
number of rows (observations) =  4
usedRowIndex: 1  2  3  4
usedColIndex: 1  2  3
Upper-left 3 columns and  4 rows:
     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3  NaN   11
row4    4    8   12
```

Now 'dat2.txt' contains the data readable with

```
> read.table("dat2.txt")
```

```
     col1 col2 col3
row1    1    5    9
row2    2    6   10
row3    3   NA   11
row4    4    8   12
```

5 Using `apply2dfo` function

The `apply2dfo` is a powerful function allowing complicated analysis of data stored in `databel` matrix. We will demonstrate the basic use of this function here. First, we will compute row and columns sums:

```
> apply2dfo(SNP, dfodata = dat2, anFUN = "sum", MAR = 2)
```

```
     [,1]
col1   10
col2  NaN
col3   42
```

```
> apply2dfo(SNP, dfodata = dat2, anFUN = "sum", MAR = 1)
```

```
     [,1]
row1   15
row2   18
row3  NaN
row4   24
```

the 'SNP' stays for current analysis variable (row or column) and allows specification of more complicated analysis, e.g.

```
> apply2dfo(SNP^2, dfodata = dat2, anFUN = "sum", MAR = 2)
```

```
     [,1]
col1   30
col2  NaN
col3  446
```

or such analysis as consecutive linear regression

```
> Y <- rnorm(4)
> apply2dfo(Y ~ SNP, dfodata = dat2, anFUN = "lm", MAR = 2)

       Estimate Std. Error  Pr(>|t|)
col1 0.08301457 0.25819760 0.7783110
col2 0.18021372 0.03377288 0.1179375
col3 0.08301457 0.25819760 0.7783110

> apply2dfo(Y ~ SNP + I(SNP^2), dfodata = dat2, anFUN = "lm", MAR = 2)

                 Estimate Std. Error  Pr(>|t|)
col1_SNP      -0.98573387  1.7666869 0.6760044
col1_I(SNP^2)  0.21374969  0.3478161 0.6491920
col2_SNP       0.77817639        NaN       NaN
col2_I(SNP^2) -0.04549716        NaN       NaN
col3_SNP      -4.40572887  7.3107611 0.6547257
col3_I(SNP^2)  0.21374969  0.3478161 0.6491920
```

Even more complicated analysis may be done by user specifying own analysis and result processing functions (see package documentation).

6 Citation

WILL BE UPDATED AT THE TIME THE PAPER IS ACCEPTED

```
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 233292  6.3     467875 12.5   407500 10.9
Vcells 118076  1.0     786432  6.0   446797  3.5
```