Strategic Data Handling for Pharmacometrics

Timothy Bergsma

Metrum Research Group, Tariffville, CT

Objectives

The relational database model [1] specifies that record storage order must be non-informative. In contrast, many pharmacometric analyses use data formats with highly informative record order. Our objective was to develop succinct, expressive, rapid data assembly techniques that respect record order.

Methods

Software was implemented within the package Mifuns (current version 4.2.2): a publicly available [2] extension of the R programming language [3]. Novel functions were developed by strategic extension of base R functions. The function stableMerge extends base merge to produce an order-stable left-join. The functions first, last, nth, and only extend base match to identify indexed singularities and repeat them at all positions in a level. The function distance analyzes positional information with respect to singularities identified by nth; derived functions before, at, and after summarize distance information. Finally, reapply extends base tapply, stretching each sub-result (using base rep) to the original length while preserving indexed order.

Conclusions

To simplify this discussion, we classify functions by the dimensionality of their values: scalar functions (e.g., mean), vector functions (e.g., match), and table functions (e.g., merge). Further, we define *processors* as functions whose value has the same dimensionality as the primary argument and *isometric functions* as those whose value has the same extent (length of first dimension) as the primary argument. Last, we define *stable functions* as processors that do not permute the first dimension more than necessary. The term is borrowed from sorting theory; in the present context, "not more than necessary" implies "not at all". Under these definitions, stableMerge is a stable isometric *table* processor, while nth (etc.) and reapply are stable isometric *vector* processors.

Stable isometric processors have strategic utility in pharmacometric data assembly, where the dominant workflow paradigm involves "building up" a table by stepwise modifications. Processors support the paradigm by returning data in the same form as input: either a variant of a table or a variant of a column. Isometric functions prevent unintentional deletions or additions. Stable functions preserve informative record order without the need for additional sorting and intermediate data objects. Stable isometric processors combine these benefits with potentially powerful effect.

Many pharmacometric data handling operations may be expressed succinctly using the functions described, all of which respect original record order. In fact, record order is central to the computational strategy of nth (etc.). In contrast, record order is computationally immaterial for reapply when used with a scalar function argument, such as mean, but will be consequential when reapply is used with "longitudinal" isometric vector processors, e.g., cumsum and rev (but not sort).

In summary, stableMerge, nth (etc.), and reapply promote strategic data handling for pharmacometrics by enabling succinct, expressive, rapid data assembly techniques that respect record order.

References

[1] Codd, E.F. The Relational Model for Database Management: Version 2 (Addison-Wesley, Reading, Mass., 1990).

[2] http://cran.r-project.org/web/packages/MIfuns

[3] http://www.r-project.org

Results

stableMerge

Pharmacometric data handling frequently requires a left-join, e.g., adding subject-level demographic variables to a table ordered by subject and time. In base R, left-joins are achieved using merge(x, y, all.x=TRUE, all.y=FALSE). However, the R help for merge clearly indicates that the value of merge may have column order, row order, and row names different from those of the primary argument (x). In fact, even the number of rows may change, if there are duplicates within matching columns of y. In contrast, stableMerge(x,y) guarantees that in its value (with respect to x) no rows are added, dropped, renamed, or reordered and that columns are not reordered. The function stableMerge deconstructs the by argument of merge, so the user must ensure that left and right columns with matching meaning have matching names, and vice versa (good general practice). Repeated matches within y are disallowed. The result is a rapid, intelligible, safe technique for left-joins that gives order-stable output.

```
 library(MIfuns)
MIfuns 4.2.2
 conc
  C SUBJ TIME CMT DV
1 1    a    0   1  8
2 0    a    0   1  1
3 0    a    0   2  2
4 0    a    1   1  5
5 0    a    1   2 10
6 0    b    0   1  2
7 0    b    0   2  4
8 0    b    1   1 10
9 0    b    1   2 20
 dem
  SUBJ SEX AGE
1    a   0  23
2    b   1  44
 stableMerge(conc,dem)
  C SUBJ TIME CMT DV SEX AGE
1 1    a    0   1  8   0  23
2 0    a    0   1  1   0  23
3 0    a    0   2  2   0  23
4 0    a    1   1  5   0  23
5 0    a    1   2 10   0  23
6 0    b    0   1  2   1  44
7 0    b    0   2  4   1  44
8 0    b    1   1 10   1  44
9 0    b    1   2 20   1  44
```

nth

Whereas the arguments and value of stableMerge are tables (class data.frame), the arguments and value of nth(x, where, within, n=1, ...) are vectors. In a table assembly context, use of nth is equivalent to a subset-and-left-join operation. The function nth returns, for each position in x, the n^{th} element of x, optionally limiting candidate elements by where and optionally breaking the evaluation across subsets, as specified by within. The argument n can be 0, returning all NA; or negative, which counts instances from the end of the vector (or subsets). If n is NA, all elements are returned. "Interlaced" levels are handled correctly. If x is missing, nth returns subscripts (useful for subsetting other variables).

Using nth, one can compute (for instance) subject-wise, compartment-wise differences of each value in a vector from some arbitrary subset member.

```
 within(
     conc,
     delta <- DV - nth(
         DV,
         where=!C,
         within=list(SUBJ, CMT),
         n=2
     )
 )
  C SUBJ TIME CMT DV delta
1 1    a    0   1  8     3
2 0    a    0   1  1    -4
3 0    a    0   2  2    -8
4 0    a    1   1  5     0
5 0    a    1   2 10     0
6 0    b    0   1  2    -8
7 0    b    0   2  4   -16
8 0    b    1   1 10     0
9 0    b    1   2 20     0
```

Other functions extend nth. The functions first and last are convenience wrappers that set n to 1 and -1, respectively. The function only is like first but returns NA if the first value is not the only value. The function distance returns subscripts less the nth subscripts, optionally considering where and within. The functions before, at, and after test whether distance is less than, equal to, or greater than zero.

reapply

Implicitly, nth applies an element-select function to cells of an indexed vector, repeating the scalar result for all elements in the cell. With reapply(x, INDEX, FUN, ...), a function is named explicitly, and need not return a scalar value. The function reapply applies a function to each cell of a vector, as specified by levels of the index. However, reapply repeats each sub-result as necessary to match the number of input elements per cell, and returns the global result as a vector in an order corresponding to the original index. The result is a highly flexible, compact mechanism for manipulation of indexed vectors.

For example, consider a table of blood pressure data. We wish to impute missing MMHG using locf, and add columns for baseline and mean, per ID and ENDP. The following strategic and conventional methodologies give identical output, but the former is more compact, expressive, and understandable.

strategic

```
 x$locf <- with(x,reapply(MMHG,INDEX=list(ID,ENDP),FUN=locf))
 x$base <- with(x,first(MMHG,where=!is.na(MMHG),within=list(ID,ENDP)))
 x$mean <- with(x,reapply(MMHG,INDEX=list(ID,ENDP),FUN=mean,na.rm=TRUE))
```

conventional

```
 x <- split(x,x[,c('ID','ENDP')])
 x <- lapply(
     x,
     function(x){
         x$locf <- locf(x$MMHG)
         x$base <- x$MMHG[!is.na(x$MMHG)][1]
         x$mean <- mean(x$MMHG,na.rm=TRUE)
         x
     }
 )
 x <- do.call(rbind,x)
 x <- x[order(x$ID,x$TIME,-x$EVID,factor(x$ENDP,levels=c('na','SYS','DIA'))),]
 rownames(x) <- NULL
 x$ENDP <- as.character(x$ENDP)

 x
   ID TIME EVID AMT ENDP MMHG locf base mean
1   1    0    2  NA  SYS  200  200  200  130
2   1    0    2  NA  DIA  110  110  110   80
3   1    0    1  10   na   NA   NA   NA  NaN
4   1   24    2  NA  SYS   NA  200  200  130
5   1   24    2  NA  DIA   NA  110  110   80
6   1   48    2  NA  SYS  120  120  200  130
7   1   48    2  NA  DIA   80   80  110   80
8   1   72    2  NA  SYS   70   70  200  130
9   1   72    2  NA  DIA   50   50  110   80
10  2    0    2  NA  SYS   NA   NA  120  127
11  2    0    2  NA  DIA   NA   NA   80   80
12  2    0    1  10   na   NA   NA   NA  NaN
13  2   24    2  NA  SYS  120  120  120  127
14  2   24    2  NA  DIA   80   80   80   80
15  2   48    2  NA  SYS  150  150  120  127
16  2   48    2  NA  DIA   90   90   80   80
17  2   72    2  NA  SYS  111  111  120  127
18  2   72    2  NA  DIA   70   70   80   80
```