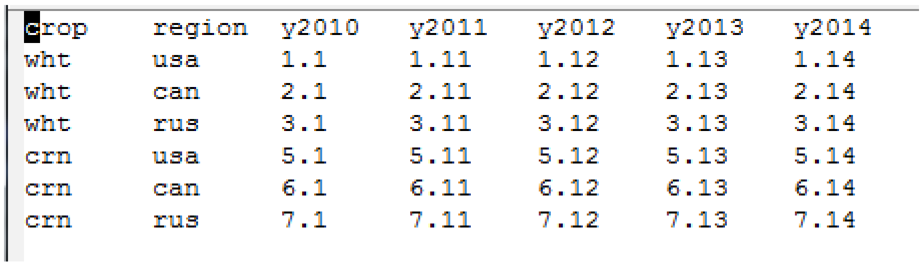
Author: Steve Dirkse
E-mail: R@gams.com
Theme: wgdx.reshape
Status: DRAFT

1 The problem

One weak spot in **gdxrrw** is the lack of support for data frames that result from calls to *read.csv* when the data is in 'wide' format. For example, the data file `sample1.dat` looks like this:

Figure 1: sample CSV data in wide format



```
crop    region  y2010   y2011   y2012   y2013   y2014
wht     usa     1.1     1.11    1.12    1.13    1.14
wht     can     2.1     2.11    2.12    2.13    2.14
wht     rus     3.1     3.11    3.12    3.13    3.14
crn     usa     5.1     5.11    5.12    5.13    5.14
crn     can     6.1     6.11    6.12    6.13    6.14
crn     rus     7.1     7.11    7.12    7.13    7.14
```

and can be read easily into an R data frame:

```
sample1 <- read.csv("sample1.dat", sep = "\t", header = T)
str(sample1)

## 'data.frame': 6 obs. of  7 variables:
##  $ crop  : Factor w/ 2 levels "crn","wht": 2 2 2 1 1 1
##  $ region: Factor w/ 3 levels "can","rus","usa": 3 1 2 3 1 2
##  $ y2010 : num  1.1 2.1 3.1 5.1 6.1 7.1
##  $ y2011 : num  1.11 2.11 3.11 5.11 6.11 7.11
##  $ y2012 : num  1.12 2.12 3.12 5.12 6.12 7.12
##  $ y2013 : num  1.13 2.13 3.13 5.13 6.13 7.13
##  $ y2014 : num  1.14 2.14 3.14 5.14 6.14 7.14
```

On inspection, we recognize that this is really data for a 3-dimensional parameter. The GAMS representation for this same data might look like this:

```
parameter prd(crop, region, time) 'production' /
wht.usa.y2010 1.1
wht.usa.y2011 1.11
* etc., etc.
crn.rus.y2013 7.13
crn.rus.y2014 7.14 /;
```

In order to properly write the data frame `sample1` to GDX via *wgdx.df*, we need to first reshape it. R provides some help here with the *reshape* or *melt* functions but even so, this can be a somewhat time-consuming and error-prone process.

2 The solution

To make reshaping easier and more reliable we will automate the process with a wrapper utility. We assume that the data is stored in an R data frame in the usual way. We also assume that the user specifies explicitly the number of dimensions she expects the GAMS data to have. While this may not be strictly necessary in all cases it adds clarity and avoids surprising results.

```
# the data file and read.csv should not change, that is basic R stuff and
# we don't want to force any change there
sample1 <- read.csv("sample1.dat", sep = "\t", header = T)
# we are really dealing with 3-dimensional data: prd(crop, region, year)
symDim <- 3
```

The are two required arguments to *wgdx.reshape*: the input data frame and the symbol dimension. In this case, the wrapper assumes that the first `symDim-1` columns are index columns, while each of the remaining columns is a data column. A data column consists of an index in the column header and data for that index below this. The indices from all the data columns are combined into the final index set (default name: `time`). It is always necessary to provide a symbol name when writing to GDX. The symbol name can be added to the data frame as an attribute, which is sometimes handy. The returned list is suitable for passing to *wgdx.lst*, either by itself or in combination with other data that need to be combined in the same GDX container.

```
sample1a <- sample1
attr(sample1a, "symName") <- "production"
lst <- wgdx.reshape(sample1a, symDim)
wgdx.lst("test1.gdx", lst)
str(lst)

## List of 4
##  $ :List of 3
##   ..$ name: chr "crop"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:2] "crn" "wht"
##  $ :List of 3
##   ..$ name: chr "region"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:3] "can" "rus" "usa"
##  $ :List of 3
##   ..$ name: chr "time"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:5] "y2010" "y2011" "y2012" "y2013" ...
##  $ :'data.frame': 30 obs. of  4 variables:
##   ..$ crop  : Factor w/ 2 levels "crn","wht": 2 2 2 1 1 1 2 2 2 1 ...
##   ..$ region: Factor w/ 3 levels "can","rus","usa": 3 1 2 3 1 2 3 1 2 3 ...
##   ..$ time  : Factor w/ 5 levels "y2010","y2011",..: 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ value : num [1:30] 1.1 2.1 3.1 5.1 6.1 7.1 1.11 2.11 3.11 5.11 ...
##   ..- attr(*, "symName")= chr "production"
```

Note that in addition to the 3-dimensional numeric production data, the list returned contains the index sets crop, region, and time defined by the data.

2.1 Writing to GDX directly

By default, *wgdx.reshape* returns a reshaped dataframe in a list. This makes it possible to combine the data returned with other data and write it all to the same GDX. However, there is some overhead in doing this. R is known for being wasteful of memory and in this case there are one or more copies of the data being made that would not be necessary if the data were written to GDX directly by the *wgdx.reshape* wrapper instead of being returned. To write to GDX directly (and more efficiently and perhaps more conveniently), simply specify the GDX name on the argument list, and the reshaped data will go directly to GDX instead of being returned in a list.

```
wgdx.reshape(sample1a, symDim, gdxName = "test2.gdx")
```

2.2 Optional arguments

The *wgdx.reshape* wrapper makes several assumptions about how it should reshape the data and what should be returned or written to GDX. These assumptions correspond to the default settings of several optional arguments discussed below.

It is possible to pass the symbol name as an argument. This argument takes precedence over the attribute specification of the symbol name.

```
lst <- wgdx.reshape(sample1a, symDim, symName = "prd")
wgdx.lst("test3.gdx", lst)
attributes(lst[[4]])$symName

## [1] "prd"
```

It is possible to specify the name of the aggregated index set created by the wrapper from the index elements taken from the data rows if the default name `time` is not suitable.

```
lst <- wgdx.reshape(sample1a, symDim, tName = "year")
lst[[3]]

## $name
## [1] "year"
##
## $type
## [1] "set"
##
## $uels
## $uels[[1]]
## [1] "y2010" "y2011" "y2012" "y2013" "y2014"
##
##
```

If you would like to include explanatory text for the symbols written to GDX you can do so using a combination of an attribute and an optional argument, as in the following example.

```
sample1b <- sample1
attr(sample1b, "symName") <- "prd_b"
attr(sample1b, "ts") <- "grain production in MMT: text for prd_b"
myText <- c("grains we produce", "aggregate regions", "crop years")
lst <- wgdx.reshape(sample1b, symDim, setNames = myText)
str(lst)
```

```
## List of 4
##  $ :List of 4
##   ..$ name: chr "crop"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:2] "crn" "wht"
##   ..$ ts  : chr "grains we produce"
##  $ :List of 4
##   ..$ name: chr "region"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:3] "can" "rus" "usa"
##   ..$ ts  : chr "aggregate regions"
##  $ :List of 4
##   ..$ name: chr "time"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:5] "y2010" "y2011" "y2012" "y2013" ...
##   ..$ ts  : chr "crop years"
##  $ :'data.frame': 30 obs. of  4 variables:
##   ..$ crop  : Factor w/ 2 levels "crn","wht": 2 2 2 1 1 1 2 2 2 1 ...
##   ..$ region: Factor w/ 3 levels "can","rus","usa": 3 1 2 3 1 2 3 1 2 3 ...
##   ..$ time  : Factor w/ 5 levels "y2010","y2011",..: 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ value : num [1:30] 1.1 2.1 3.1 5.1 6.1 7.1 1.11 2.11 3.11 5.11 ...
##   ..- attr(*, "symName")= chr "prd_b"
##   ..- attr(*, "ts")= chr "grain production in MMT: text for prd_b"
```

In some cases the index sets used to declare and work with the data in `sample1` may already exist or may be different from those implied by `sample1` (e.g. if `sample1` only contains a subset of the data). If so, the extraction of these sets can be suppressed by setting *setsToo=FALSE*.

```
lst <- wgdx.reshape(sample1a, symDim, setsToo = FALSE)
str(lst)

## List of 1
##  $ :'data.frame': 30 obs. of  4 variables:
##   ..$ crop  : Factor w/ 2 levels "crn","wht": 2 2 2 1 1 1 2 2 2 1 ...
##   ..$ region: Factor w/ 3 levels "can","rus","usa": 3 1 2 3 1 2 3 1 2 3 ...
##   ..$ time  : Factor w/ 5 levels "y2010","y2011",..: 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ value : num [1:30] 1.1 2.1 3.1 5.1 6.1 7.1 1.11 2.11 3.11 5.11 ...
##   ..- attr(*, "symName")= chr "production"

length(lst)

## [1] 1
```

2.3 Reordering the data

By default, the first `symDim-1` columns of the input data frame are used as the first index columns of the result, while the final index column of the result is the aggregation of the column headers from the data columns. This can be changed by using the *order* argument.

If specified, *order* must be a vector of length *symDim*. The value in *order[k]* specifies what column of the input data frame to use as the *k'th* index column of the output. To indicate which index position takes the aggregated data column headers, use a non-positive value or '*' for *order[k]*.

```
# this duplicates the default ordering: crop, region, time
lst <- wgdx.reshape(sample1a, symDim, order = c(1, 2, -1))
wgdx.lst("testDefaultOrder.gdx", lst)
str(lst)

## List of 4
##  $ :List of 3
##   ..$ name: chr "crop"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:2] "crn" "wht"
##  $ :List of 3
##   ..$ name: chr "region"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:3] "can" "rus" "usa"
##  $ :List of 3
##   ..$ name: chr "time"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:5] "y2010" "y2011" "y2012" "y2013" ...
##  $ :'data.frame': 30 obs. of  4 variables:
##   ..$ crop  : Factor w/ 2 levels "crn","wht": 2 2 2 1 1 1 2 2 2 1 ...
##   ..$ region: Factor w/ 3 levels "can","rus","usa": 3 1 2 3 1 2 3 1 2 3 ...
##   ..$ time  : Factor w/ 5 levels "y2010","y2011",..: 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ value : num [1:30] 1.1 2.1 3.1 5.1 6.1 7.1 1.11 2.11 3.11 5.11 ...
##   ..- attr(*, "symName")= chr "production"
```

```
# this orders the output as prd(year,crop,region)
lst <- wgdx.reshape(sample1a, symDim, tName = "year", symName = "prd",
    order = c(-1, 1, 2))
wgdx.lst("testYearCropRegion.gdx", lst)
str(lst)

## List of 4
##  $ :List of 3
##   ..$ name: chr "year"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:5] "y2010" "y2011" "y2012" "y2013" ...
##  $ :List of 3
##   ..$ name: chr "crop"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:2] "crn" "wht"
##  $ :List of 3
##   ..$ name: chr "region"
##   ..$ type: chr "set"
##   ..$ uels:List of 1
##   .. ..$ : chr [1:3] "can" "rus" "usa"
##  $ :'data.frame': 30 obs. of  4 variables:
##   ..$ year  : Factor w/ 5 levels "y2010","y2011",..: 1 1 1 1 1 1 2 2 2 2 ...
##   ..$ crop  : Factor w/ 2 levels "crn","wht": 2 2 2 1 1 1 2 2 2 1 ...
##   ..$ region: Factor w/ 3 levels "can","rus","usa": 3 1 2 3 1 2 3 1 2 3 ...
##   ..$ value : num [1:30] 1.1 2.1 3.1 5.1 6.1 7.1 1.11 2.11 3.11 5.11 ...
##   ..- attr(*, "symName")= chr "prd"
```