A note on 'Dasst' package for working with 'DSSAT-CSM' files

Homero Lozza

October 31, 2017

1 Introduction

The 'DSSAT-CSM' [1] writes many of its variable values in text files using fixed width format (FWF). These files typically have `.OUT` extension. In addition, input data must also be formatted in columns of fixed width [2].

The **Dasst** package provides 'DSSAT' users with methods for easy accessing the values on 'DSSAT' files. It enables simulated treatments to be prepared and studied with numerous tools available in **R** for statistical and graphical analyses.

2 Processing a `.OUT` file

The first step is to read the required `.OUT` file and to store it in memory as an S4 object of class `Dasst`. We delivered some example files on the `extdata` directory of the **Dasst** installation. In order to perform the following examples, we will read the `PlantGro.OUT` file which contains the daily plant growth for 3 treatments repeated over 10 years. Runs ranging from 1 to 10 have plant output data for treatment 1 which simulates the growth of a maize from 1970 to 1979 (south hemisphere). Treatment 2 is analogous to the previous one, but adds a fertilization with 50kg/ha of urea. Lastly, treatment 3 adds a fertilization with 100kg/ha of urea.

The `read.dssat` function can be invoked passing a vector of characters with the file name/names to be opened. Also other arguments are accepted to restrict the number of fields to be retrieved. See help for more details.

```
library(Dasst)
dssatfile <- system.file("extdata","PlantGro.OUT",
                         package="Dasst")
plantgro <- read.dssat(dssatfile)
```

The same data set is available and can be load by means of

```
data(plantGrowth)
```

The S4 object stores internally the values immediately bellow each header line of the text file as a `data.frame`. Each `data.frame` is identified with a table.

```
length(plantGrowth)

## [1] 30
```

In this case, we count with 30 tables. This means that this was the number of headers found in the text file.

The `show` method displays the contents of the first table stored in an object of class `Dasst`. The `summary` method gives a brief report of the information stored in the `Dasst` object, and gives the number of fields and records for each table.

```
summary(plantGrowth)

## * Object of class =  Dasst
## * Files    =  1
## * Sections =  30
## * Tables   =  30
## Table  1 :  45 fields and 131 records
## Table  2 :  45 fields and 112 records
## Table  3 :  45 fields and 124 records
## Table  4 :  45 fields and 133 records
## Table  5 :  45 fields and 129 records
## Table  6 :  45 fields and 122 records
## Table  7 :  45 fields and 127 records
## Table  8 :  45 fields and 120 records
## Table  9 :  45 fields and 117 records
## Table  10 :  45 fields and 123 records
##
## ... Print limited to the first  10  tables.
## * Total records =  3714
```

Each table contents is accessed by the `[[` method. Specific values can be set by rows, column names, or a combination of both. For example, the value for the `"YEAR"` field on row `1` corresponding to table `1` is

```
plantGrowth[[1]][1,"YEAR"]

## [1] 1970
```

The table orders are suitable for performing operations, and we can find them searching for patterns within the filename, section and/or header parts belonging to each table. Also, we can print a brief summary of the results.

```
nrun <- searchAncillary(plantGrowth, secKey="run[[:space:]]*3",
                        ignore.case=TRUE)
nrun

## [1]  3 30

getAncillary(plantGrowth, nrun)

## * Showing ancillary data for selected table orders:
## Orders:             Files:              Sections:           Columns:
##                  3  PlantGro.OUT        *RUN   3        ... YEAR DOY DAS DA ...
##                 30  PlantGro.OUT        *RUN  30        ... YEAR DOY DAS DA ...
##
## For more, ancillary_object[[<name>]]; <name>: orders|files|sections|columns.
```

2.1 Dates processing

The conversion from year (`YEAR`) and day of the year (`DOY`) is perform by `addDate<-` method. It stores dates as `Date` objects in a new column named `date_YEAR_DOY`.

```
addDate(plantGrowth) <- ~ YEAR + DOY
plantGrowth[[1]][1:3,c("YEAR","DOY","date_YEAR_DOY")]

##   YEAR DOY date_YEAR_DOY
## 1 1970 288    1970-10-15
## 2 1970 289    1970-10-16
## 3 1970 290    1970-10-17
```

This turns very simply the sketch of evolution curves.

```
plot(plantGrowth[[1]][,"date_YEAR_DOY"],plantGrowth[[1]][,"LAID"])
```

2.2 Mean operation

Several operations can be performed over the whole data set or restricted to a subset. We could be interested on computing the mean, variance, maximum, or minimum values per treatment. In the following example, we will compute the mean values of stem, leaf and grain weight for each day after planting (`DAP`) for treatment 1 that spans runs from 1 to 10:

```
from01to10 <- gatherTables(plantGrowth[1:10], c("DAP"),
                           c("SWAD","LWAD","GWAD"), mean)
lastRow <- nrow(from01to10)
from01to10[(lastRow-5):lastRow,]
```

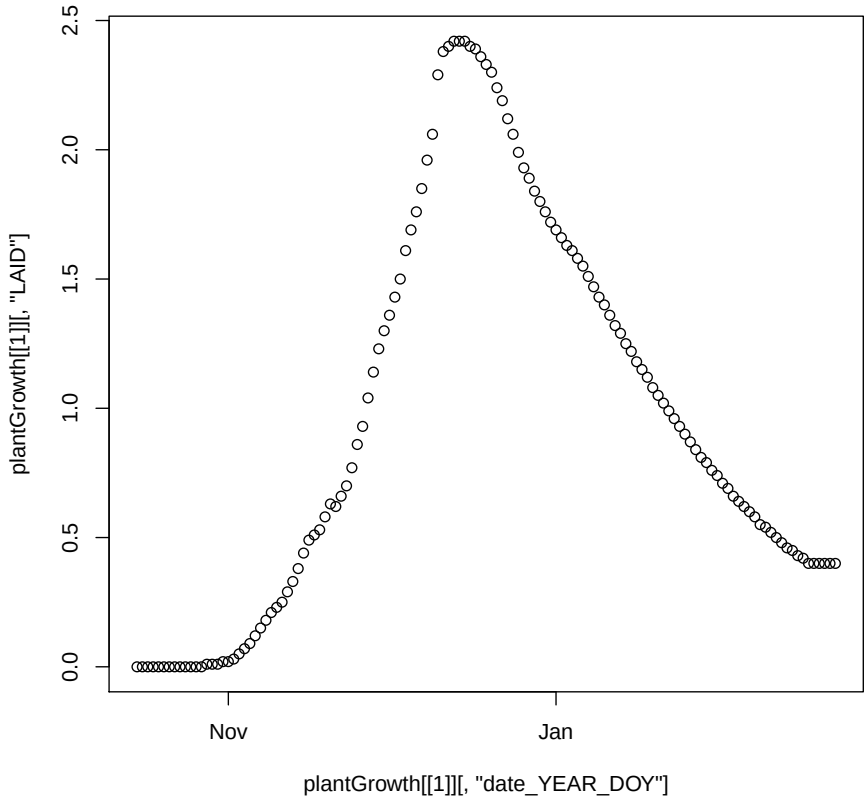

Figure 1: Daily evolution of `LAID`.

```
##     DAP SWAD LWAD GWAD
## 107 106 2224 2112 3301
## 108 107 2194 2112 3423
## 109 108 2173 2112 3530
## 110 109 2152 2112 3633
## 111 110 2125 2112 3736
## 112 111 2096 2112 3839
```

The result is stored in a `data.frame` which can be manipulated and plotted with standard tools:

```
plot(SWAD + LWAD + GWAD ~ DAP, data=from01to10)
```

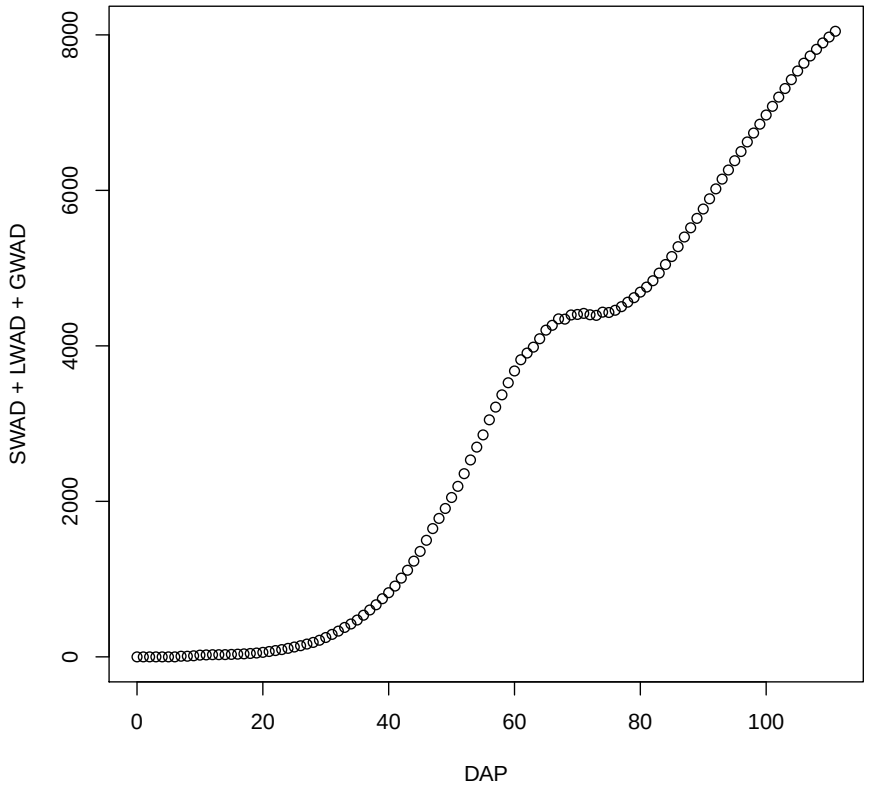

Figure 2: Mean daily evolution of the sum of stem, leaf and grain weight.

2.3 Yield plots

Yield may be retrieved from the last row for each run of the `PlantGro.OUT` file. However, we will read the `Summary.OUT` file available from the `extdata` directory in the **Dasst** installation.

```
smmyfile <- system.file("extdata", "Summary.OUT", package="Dasst")
smmy <- read.dssat(smmyfile)
summary(smmy)

## * Object of class =  Dasst
## * Files    =  1
## * Sections =  1
## * Tables   =  1
## Table  1 :  52 fields and 30 records
## * Total records =  30
```

From the summary information we notice that the `smmy` object contains only 30 records gather in table 1. This corresponds to the 10 years of simulations for each of the 3 treatments simulated.

```
boxplot(HWAM ~ TRNO, data=smmy[[1]])
```

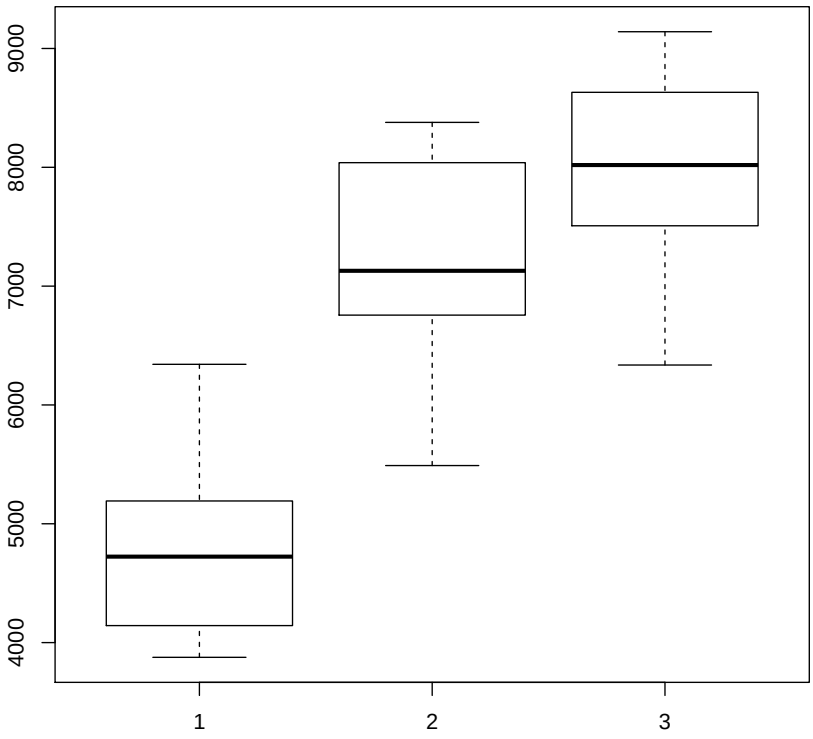

Figure 3: Yield box plot. Treatments; 1: without fertilization, 2: with 50kg/ha of urea, 3: with 100kg/ha of urea.

3 Stacking files

On certain occasions, you could be interested in time series that span across several tables. For example, when 'DSSAT-CSM' is executed in sequence mode, soil water values corresponding to the *N run* go on *N+1 run*. Here, we will address the generation of time series from `.WTH` files.

We will read the `.WTH` files available from the `extdata` directory in the **Dasst** installation.

```
wthPath <- paste(find.package("Dasst"),"extdata",sep="/")
wthFiles <- list.files(path=wthPath, pattern="WTH",
                       full.names=TRUE)
wth <- read.dssat(wthFiles)
summary(wth)

## * Object of class =  Dasst
## * Files    =  11
## * Sections =  1
## * Tables   =  22
## Table  1 :  8 fields and 1 records
## Table  2 :  5 fields and 365 records
## Table  3 :  8 fields and 1 records
## Table  4 :  5 fields and 365 records
## Table  5 :  8 fields and 1 records
## Table  6 :  5 fields and 366 records
## Table  7 :  8 fields and 1 records
## Table  8 :  5 fields and 365 records
## Table  9 :  8 fields and 1 records
## Table  10 :  5 fields and 365 records
##
## ... Print limited to the first  10  tables.
## * Total records =  4029
```

From the summary information we notice that the `wth` object contains 22 tables. We found 365 or 366 records on half of them according to the year. Interestingly, the small tables with only 1 record stores the climatic information which in the `.WTH` files looks like

INSI	LAT	LONG	ELEV	TAV	AMP	TMHT	WMHT
	-34.071	-60.304	-99	16.5	12.0	2.0	-99

In order to obtain the full time series, we will stack only the even table numbers

```
wthSeries <- stackTables(wth[seq(from=2, to=22, by=2)])
```

The result is available as `data.frame`.

```
plot(TMAX ~ as.Date(as.character(DATE),format="%y%j"),
     wthSeries)
```

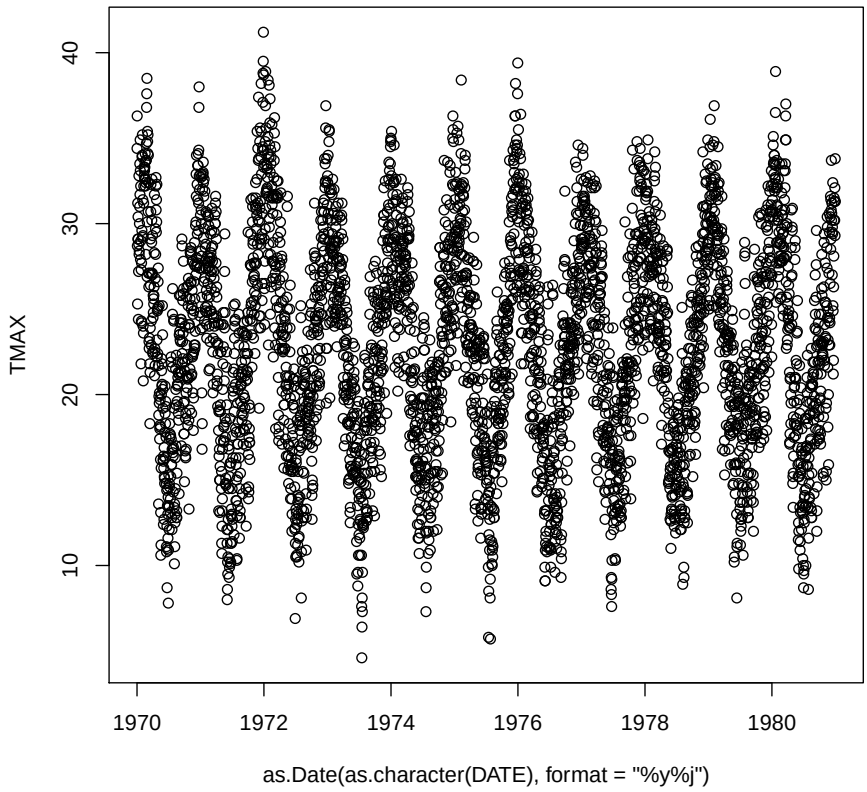

Figure 4: Daily maximum temperature series.

4 Input file edition

This section gives a brief idea of the potential of `write.dssat` method for the edition of DSSAT-CSM input files. Further details can be found in other vignette aimed at automatic calibration and optimization of 'DSSAT-CSM' input files.

We will read the experimental file `SANT7001.MZX` and we will replace the fertilization doses. Because this file will be replaced, we make a copy on a temporary directory.

```
santfile <- system.file("extdata", "SANT7001.MZX",
                        package="Dasst")
ffn <- paste(tempdir(), "SANT7001.MZX", sep="/")
file.copy(santfile, ffn)

## [1] TRUE

sant <- read.dssat(ffn)
sant[[9]]

##   F FDATE   FMCD   FACD FDEP FAMN FAMP FAMK FAMC FAMO FOCD FERNAME
## 1 1 70288  FE005  AP003   10   50    0    0    0    0  -99     -99
## 2 2 70288  FE005  AP003   10  100    0    0    0    0  -99     -99

sant[[9]][,"FAMN"] <- c(60,120)
sant[[9]]

##   F FDATE   FMCD   FACD FDEP FAMN FAMP FAMK FAMC FAMO FOCD FERNAME
## 1 1 70288  FE005  AP003   10   60    0    0    0    0  -99     -99
## 2 2 70288  FE005  AP003   10  120    0    0    0    0  -99     -99

write.dssat(sant, ffn)
```

The `SANT7001.MZX` was rewritten and the original file was saved as `SANT7001.MZX.bak`.

5 Known issues

Sections that do not contain values are skipped, and there will not be tables representing these names. Moreover, comments are also skipped and will not be found in their corresponding files if `write.dssat` method is applied. In addition, some details that may be found between the section name and the variables header will be skipped. In general, verbose reduction does not affect the program behavior because merely comments are erased.

The column widths are computed from the space reserved in header lines for each field. Each variable header is assumed to be right justified. The column widths are computed as the number of characters (including blanks) spanning from the last character of the previous word up to the end of the following variable name. Thus, header lines containing variable names with spaces in between are misunderstand. Eventually, headers from such a file should be edited and spaces should be replaced by underscores.

The type of value is detected automatically. Fields are assumed numeric if only figures, dots, pluses or minuses are found. Otherwise they are considered as a character strings. Sometimes, the columns for character fields are left justified. This adds some difficulties in the column width detection. To fix some misalignment, extend the variable name with underscores up to one space before the beginning of the following column header.

6 Conclusion

In conclusion, this package tends to simplify the post processing of 'DSSAT' simulated values stored in `.OUT` files offering methods that expose these data as belonging to a collection of `data.frame` objects that can be thought like tables.

References

[1] Jones, J.W., G. Hoogenboom, C.H. Porter, K.J. Boote, W.D. Batchelor, L.A. Hunt, P.W. Wilkens, U. Singh, A.J. Gijsman, and J.T. Ritchie. 2003. DSSAT Cropping System Model. European Journal of Agronomy 18:235-265

[2] G.Y. Tsuji, G. Uehara and S. Balas (eds.). 1994. DSSAT v3 (Volume 2). University of Hawaii, Honolulu, Hawaii.