Computers and Electronics in Agriculture 172 (2020) 105325

Contents lists available at ScienceDirect

Computers and Electronics in Agriculture

journal homepage: www.elsevier.com/locate/compag

A comprehensive R interface for the DSSAT Cropping Systems Model

Phillip D. Alderman

Department of Plant and Soil Sciences, Oklahoma State University, 371 Agricultural Hall, Stillwater, OK 74078, USA

A B S T R A C T

The Decision Support System for Agrotechnology Transfer Cropping Systems Model (DSSAT-CSM) is a widely used modeling system. The `DSSAT` R package was developed to provide tools that would facilitate preparing required model inputs, executing simulations, and processing and analyzing outputs for DSSAT-CSM. This application note demonstrates the use of this new package for building reproducible crop modeling workflows using the DSSAT-CSM system. Example workflows are provided for modifying values in input data files (soil, weather, and experimental details), running simulations, reading simulated output, and creating publication-quality visualizations of observed and simulated data. The `DSSAT` R package provides basic tools that when combined with other R packages will facilitate developing robust, reproducible, scientific modeling workflows.

1. Introduction

The Decision Support System for Agrotechnology Transfer Cropping Systems Model (DSSAT-CSM; Jones et al., 2003) is a widely used crop modeling system with an estimated 2,500 users across 100 countries worldwide (Koo, 2016). The standard user training demonstrates the use of the DSSAT Shell, a Windows-based graphical user interface (GUI) with various utilities for preparing input files, running simulations, and summarizing output. This interface greatly improves the accessibility of the DSSAT-CSM for beginning users. However, most advanced users of DSSAT-CSM have developed ad hoc scripts in various languages/software environments (e.g. R, Python, SAS) to automate various stages in their analysis (J.W. White, personal communication, October 17, 2019). While an ad hoc approach may be sufficient for many applications, a coherent framework for developing modeling workflows would improve the transparency, reproducibility, and productivity of crop modeling research. With such a framework, researchers would save time in the immediate term in the generation and processing of files and in the long term by making their scripts easier to understand by others and themselves retrospectively.

There have been several attempts to provide frameworks for building crop modeling workflows. For example, the `apsimr` package (Stanfill, 2015) was developed as an interface to the Agricultural Production Systems sIMulator (APSIM; Holzworth et al., 2014), a crop modeling system similar to DSSAT-CSM. The package includes functions to create, edit, and run APSIM simulations and analyze outputs from R. Similarly, `pyDSSAT` is a package that was developed to facilitate the use of DSSAT-CSM within a Python workflow (He et al., 2015). The package provides command line interface and GUI tools for manipulating simulation batch files and crop management input files, running DSSAT-CSM simulations and analyzing model outputs. Likewise, `jDSSAT` (Abreu Resenes et al., 2019) is a JavaScript Module that was developed to eventually replace the existing DSSAT Shell. The long-term goal of the effort is to provide support for all input and output types for DSSAT-CSM, but its current implementation has capabilities similar to those of `pyDSSAT`. Within the R ecosystem, the `Dasst` package (Lozza, 2017) provides tools to simplify the post-processing of output files for DSSAT-CSM, although no tools are provided for manipulating input files or running the model. While each of the above examples are valuable contributions, all are limited in scope and none constitute a generic framework for developing full crop modeling workflows with DSSAT-CSM. A generic framework would need to provide capabilities for manipulating the full range of required model inputs (e.g. crop management, soil properties, weather data, and genotype-specific parameters), running simulations and post-processing model outputs. Furthermore, none of these frameworks leverage the `tidyverse`, a set of R packages developed to enhance transparency and reproducibility of analysis based on a common design philosophy, grammar, and set of data structures (Wickham, 2019). The `DSSAT` R package was developed to provide tools consistent with `tidyverse` principles that would facilitate preparing required model inputs, executing simulations, and processing and analyzing outputs for DSSAT-CSM. This application note demonstrates the use of the new `DSSAT` R package for building reproducible crop modeling workflows with DSSAT-CSM.

2. Installing and loading the `DSSAT` package

The `DSSAT` package source code is hosted in an open-source project on Github (https://github.com/palderman/DSSAT). Source Code 1 provides example R code for installing the `DSSAT` package from either the Github source code repository using the `install_github()` function or from the Comprehensive R Archive Network (CRAN; https://cran.r-project.org) using

E-mail address: phillip.alderman@okstate.edu.

https://doi.org/10.1016/j.compag.2020.105325
Received 15 November 2019; Received in revised form 24 February 2020; Accepted 27 February 2020
Available online 20 March 2020

the `install.packages()` function. Installing from the Github source code repository requires installation of the `devtools` R package (Wickham et al., 2019). Either option should install `DSSAT` and any required dependencies not already installed on the system. Once package installation is complete, the package can be loaded using the `library()` function (Source Code 1). Full use of the `DSSAT` package requires an installation of DSSAT-CSM, which can be obtained from the DSSAT Foundation (https://dssat.net/) or by compiling it from source code (https://github.com/DSSAT/dssat-csm-os). When the `DSSAT` package is loaded, it attempts to locate the local DSSAT-CSM installation and identify the proper executable name. It then prints a start up message indicating what file path was found (if any) and prompting the user to reset the path (by setting the value of the `DSSAT.CSM` option variable) if the located file path is incorrect. Source Code 1 shows two examples for setting the file path to the DSSAT-CSM executable. The first is an example path for a Windows installation. The second example is compatible with a Unix-style operating system (e.g. macOS, Linux, etc). The following sections illustrate use of the most important functions available in the package. However, a complete list of functions can be found in the reference manual on CRAN (https://cran.r-project.org/package=DSSAT).

Source Code 1. Example code for installing and loading the DSSAT R package.

```
# Install DSSAT package from Github source using devtools package
devtools::install_github('https://github.com/palderman/DSSAT')
# Install DSSAT package from CRAN
install.packages('DSSAT')
# Load the DSSAT package
library(DSSAT)
# Example setting DSSAT-CSM path for Windows operating system
options(DSSAT.CSM = 'C:\\DSSAT47\\DSCSM047.EXE')
# Example setting DSSAT-CSM path for Unix-style operating system
options(DSSAT.CSM = '/DSSAT47/dscsm047')
```

3. Modifying DSSAT files

The `DSSAT` package implements a set of functions for reading and writing standard DSSAT file formats including files for cultivar (*.CUL), ecotype (*.ECO), soil (*.SOL), weather (*.WTH), experiment details (FileX), seasonal observed data (FileA), and time-series observed data (FileT). As an example, the function `read_sol()` reads soil profiles from the standard DSSAT soil file (*.SOL) format. Source Code 2 shows the use of this function within an example workflow that creates a new soil profile from an existing one and appends it to an existing soil file. The first statement reads the entire contents of the soil file `SOIL.SOL`, while the second statement reads only the profile identified by the code `IB00000001`. The output of this function is an object of class `DSSAT_tbl`, which is an extension of the `tibble` (itself an extended version of the basic data frame with enhanced functionality defined in the `tibble` package; Wickham and Grolemund, 2017; Müller and Wickham, 2019), with additional attributes used internally to store information about the original format of the file from which the data came. Some of the original data are converted into list-columns due to the one-to-many relationship between whole-profile and layer-specific data. For example, properties such as albedo (SALB) or runoff curve number (SLRO) have a single value for each profile, but other properties, such as saturation volumetric soil water (SSAT) or bulk density (SBDM), have values for each individual layer within the profile. Storing the layer-specific data as list-columns in the output from `read_sol()` facilitates reading and combining multiple soil profiles into a single combined tibble.

As an example, suppose one wanted to calculate a new value for SSAT as 95% of pore space estimated from SBDM. One could perform this calculation and replace the former values using the third statement in Source Code 2. For readers unfamiliar with the `tidyverse`-style of R programming (http://tidyverse.org; Wickham, 2017), this example uses the `%>%` pipe operator to pass the output from one line to the first argument of the function on the following line. Thus, the `single_profile` tibble is passed to `mutate()`, in which the `PEDON` column is assigned the code `IBNEW00001` and `SSAT` column is assigned the new values calculated from SBDM. This example and all following examples presuppose that the `tidyverse` package has been loaded using the `library()` function. An alternative formulation that does not use `tidyverse`-style coding is provided just below (Source Code 2). Once these changes have been made, the new profile can be appended to the existing `SOIL.SOL` by calling the function `write_sol()` with the `append` argument set to `TRUE` (the default value), as shown in the fifth statement in Source Code 2. The `write_sol()` can also be used to write a new soil file or overwrite an existing soil file by setting `append` to `FALSE`. Thus, care should be taken to avoid unintentional loss of data.

Source Code 2. Example code for reading, modifying and writing out DSSAT soil data.

```
# Reading all profiles in a file
all_profiles <- read_sol('SOIL.SOL')
# Reading a single profile
single_profile <- read_sol('SOIL.SOL',id_soil = 'IB00000001')
# Renaming the profile and replacing SSAT with new values
#     calculated from SBDM using tidyverse-style coding
new_profile <- single_profile %>%
  mutate(PEDON='IBNEW00001',
         SSAT=0.95*(2.65-SBDM)/2.65)
# Renaming the profile and replacing SSAT with new values
#     calculated from SBDM without using tidyverse-style coding
new_profile <- single_profile
new_profile$PEDON[1] <- 'IBNEW00001'
new_profile$SSAT[[1]] <- 0.95*(2.65-single_profile$SBDM[[1]])/2.65
# Appending new profile to SOIL.SOL
write_sol(new_profile,'SOIL.SOL',append=TRUE)
```

Weather data can also be imported into R in a similar way using the `read_wth()` function. The output of this function is a tibble containing the daily weather data from the DSSAT format weather file (*.WTH). The tibble also contains an attribute called `GENERAL` in which the general information about the site is stored including, among other details, the long-term average temperature (TAV) and monthly temperature amplitude (AMP). Supposing one had a directory of weather files from multiple years at the same location that were missing the TAV and AMP values, one could calculate these values from the daily data, assign them to the `GENERAL` attribute for each year, and then re-write the weather data with the new TAV and AMP values. An example workflow for this process is provided in Source Code 3. Variations of this workflow could be used to modify values within daily weather data as well as to fill missing-data gaps or combine variables from different data sources.

Source Code 3. Example workflow for modifying the values for long-term average temperature (TAV) and monthly temperature amplitude (AMP) within a set of DSSAT weather files (*.WTH).

```
# Generate a list of the weather files
wth_file_list <- list.files(pattern='.WTH')
# Read all weather files into a list of tibbles
all_wth <- wth_file_list %>%
  map(read_wth)
# Combine all years into a single tibble for summary calculations
combined_wth <- all_wth %>%
  bind_rows()
# Calculate long-term average temperature (TAV)
tav <- combined_wth %>%
  summarize(TAV=mean((TMAX+TMIN)/2))
# Calculate monthly temperature amplitude (AMP)
amp <- combined_wth %>%
  # Extract month from DATE column
  mutate(month = month(DATE)) %>%
  # Group data by month
  group_by(month) %>%
  # Calculate monthly means
  summarize(monthly_avg = mean((TMAX+TMIN)/2)) %>%
  # Calculate AMP as half the difference between minimum and
  #     maximum monthly temperature
  summarize(AMP = (max(monthly_avg)-min(monthly_avg))/2)
# Generate new general information table
general_new <- all_wth[[1]] %>% # use first year as template
  # Extract GENERAL table
  attr('GENERAL') %>%
  # Replace TAV and AMP with new values
  mutate(TAV=tav$TAV,
         AMP=amp$AMP)
# Store new general information table within each year
for(i in 1:length(all_wth)){
  # Replace general information table
  attr(all_wth[[i]],'GENERAL') <- general_new
}
# Overwrite previous weather files with modified weather data
for(i in 1:length(all_wth)){
  # Write weather file i
  write_wth(all_wth[[i]],wth_file_list[i])
}
```

The experiment details file format (FileX) is one of the most complex of the DSSAT file formats because it contains a tree-like structure with multiple tables of data that are connected by a combination of one-to-one and one-to-many relationships. At present, no attempt has been made within the DSSAT package to construct a unified relational data structure. Thus, the output of the `read_filex()` function is a named list of tibbles each element of which corresponds to a section of the FileX. The names of the list correspond to the section names of the FileX. An example workflow for adding an additional irrigation event to the `IRRIGATION AND WATER MANAGEMENT` section of a FileX is given in Source Code 4. The function `read_filex()` works similarly to the other `read_*()` functions already discussed. In the second statement, a conditional mutate function `mutate_cond()` (provided by the `DSSAT` package) is used to modify only rows that meet the conditions provided in the second argument. In this case, only rows where `I` equals 1 will be modified. Due to the one-to-many relationship between irrigation level (`I`) and the application details (`IDATE`, `IROP`, and `IRVAL`), these details are stored as list-columns, hence the data for the new event must be appended using the concatenate function `c()`. The final statement in

Source Code 4 uses `write_filex()` to write out the modified experiment details using the same file name as the original file. By using the same name the original file will be replaced by the new file. If this behavior is not desired, a different file name for the FileX may be provided.

Source Code 4. Example workflow for adding another irrigation event to an existing DSSAT experiment details file (FileX).

```
# Read in original FileX
file_x <- read_filex('KSAS8101.WHX')
# Add an additional 60 mm irrigation event on 4 May 1982
file_x$`IRRIGATION AND WATER MANAGEMENT` <-
  # Extract the original IRRIGATION AND WATER MANAGEMENT section
  file_x$`IRRIGATION AND WATER MANAGEMENT` %>%
  # Modify the IDATE, IROP, and IRVAL columns only where I equals 1
  mutate_cond(I==1,
              IDATE = c(IDATE,as.POSIXct('1982-05-04')),
              IROP  = c(IROP,"IR001"),
              IRVAL = c(IRVAL,60))
# Overwrite original FileX with new values
write_filex(file_x,'KSAS8101.WHX')
```

Although space considerations preclude providing examples for all file types, similar workflows could be constructed for other file types using the corresponding functions for reading/writing files for cultivar (`read_cul()` and `write_cul()`), ecotype (`read_eco()` and `write_eco()`), FileA (`read_filea()` and `write_filea()`), and FileT (`read_filet()` and `write_filet()`).

4. Running simulations and summarizing output

In addition to modifying input files, the DSSAT package also contains functions for generating simulation batch files, running the model, and reading simulated output. Once the option variable `DSSAT.CSM` has been set (see Section 2), the user can generate a simulation batch file as illustrated in the third and fourth statements in Source Code 5. In the third statement, the user constructs a data frame or tibble with all the necessary columns specified including, among other details, the FileX name and treatment levels to be run. In the fourth statement, the user specifies as many of the columns as are needed to uniquely specify the set of simulations and the remaining columns will be filled with default values. If the `file_name` argument is not specified, the function will attempt to construct a file name based on the current value of `DSSAT.CSM`. Once the batch file has been generated, the model can be run using the `run_dssat()` function. Once simulations have completed, the simulated output can be read using the `read_output()` function as is demonstrated in Source Code 5.

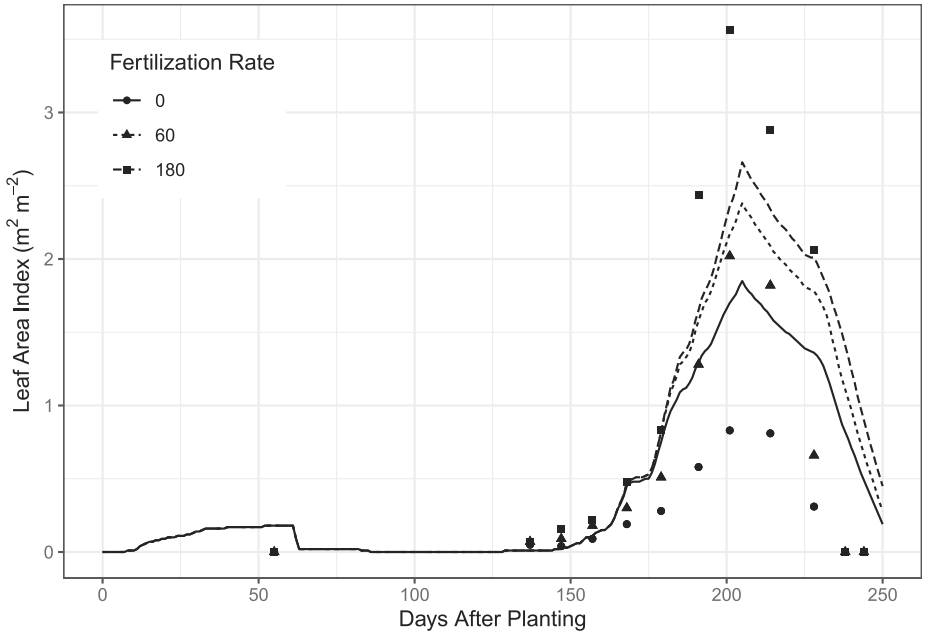

Fig. 1. Output of code shown in Source Code 6 showing observed (points) and simulated (lines) leaf area index over time for 0, 60, and 180 kg N ha^{-1} fertilization rates.

Source Code 5. Example workflow for generating a batch file, running the DSSAT-CSM model, and reading seasonal summary output.

```
# Generate a DSSAT batch file using a tibble
tibble(FILEX='KSAS8101.WHX', TRTNO=1:6, RP=1, SQ=0, OP=0, CO=0) %>%
  write_dssbatch()
# Generate a DSSAT batch file with function arguments
write_dssbatch(filex='KSAS8101.WHX',trtno=1:6)
# Run DSSAT-CSM
run_dssat()
# Read seasonal output file
smry <- read_output('Summary.OUT')
```

The `read_output()` function can also be used to read daily simulated output and generate publication-quality graphics when combined with functions from the `ggplot2` package (Wickham, 2016) as shown in Source Code 6. The first statement reads in the simulated output, converts treatment number (TRNO) to a discrete factor, and filters the output to include only treatments 4 to 6. The second statement reads in observed data from FileT format and subsets to the corresponding treatments. The final statement builds a publication-quality plot using the simulated and observed datasets, the output of which is shown in Fig. 1. Further explanation of the functions used to construct the plot can be found in the `ggplot2` documentation (Wickham, 2016).

Source Code 6. Example workflow for reading daily simulated output and generating graphics using ggplot2.

```
# Read daily simulated plant growth output
pgro <- read_output('PlantGro.OUT') %>%
  # Filter to treatments 4 to 6
  filter(TRNO %in% 4:6) %>%
  # Convert TRNO to a factor and rename to Fertilization Rate
  mutate(`Fertilization Rate`=factor(TRNO,labels=c(0,60,180)))
# Read time-series observed plant growth data from FileT
filet <- read_filet('KSAS8101.WHT') %>%
  # Filter to treatments 4 to 6
  filter(TRNO %in% 4:6) %>%
  # Convert TRNO to a factor and rename to Fertilization Rate
  mutate(`Fertilization Rate`=factor(TRNO,labels=c(0,60,180))) %>%
  # Add days after planting (DAP) to observed data
  left_join(select(pgro,DATE,DAP))
# Construct a combined plot with simulated and observed data
ggplot(data=pgro,aes(x=DAP,y=LAID,linetype=`Fertilization Rate`))+
  # Add a line plot for simulated data
  geom_line()+
  # Add observed data as points
  geom_point(data=filet,aes(shape=`Fertilization Rate`))+
  # Add a custom y-axis label with units
  ylab(expression(Leaf~Area~Index~"("*m^2~m^{-2}*")"))+
  # Add a custom x-axis label
  xlab("Days After Planting")+
  # Set color theme to black and white
  theme_bw()+
  # Reposition legend
  theme(legend.position=c(0.15,0.8))
```

5. Summary and future directions

In summary, the `DSSAT` R package provides basic functions for reading and writing input files, executing simulations, and reading simulated output files for DSSAT-CSM. These functions can be combined with other R packages to develop robust, reproducible, scientific modeling workflows. The current version of the package provides a foundation for further development of higher-level functionality such as conducting automated sensitivity analysis and parameter estimation, filling gaps in weather data, and estimating soil parameters from pedotransfer functions. Future developments for the package might also include improving the interface for manipulating FileXs, speeding up read and write operations and extending capabilities to include reading and writing species parameter files.

CRediT authorship contribution statement

Phillip D. Alderman: Conceptualization, Methodology, Software, Writing - original draft, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the USDA National Institute of Food and Agriculture, Hatch project OKL03023.

References

de Abreu Resenes, J., Pavan, W., Hölbig, C.A., Fernandes, J.M.C., Shelia, V., Porter, C., Hoogenboom, G., 2019. jDSSAT: A JavaScript module for DSSAT-CSM integration. SoftwareX 10, 100271. https://doi.org/10.1016/j.softx.2019.100271.

He, X., Peng, L., Sun, H., 2015. pyDSSAT documentation release 1.0. http://xiaoganghe.github.io/pyDSSAT/doc/_downloads/pyDSSATdoc.pdf (accessed February 12, 2020).

Holzworth, D.P., Huth, N.I., deVoil, P.G., Zurcher, E.J., Herrmann, N.I., McLean, G., Chenu, K., van Oosterom, E.J., Snow, V., Murphy, C., Moore, A.D., Brown, H., Whish, J.P., Verrall, S., Fainges, J., Bell, L.W., Peake, A.S., Poulton, P.L., Hochman, Z., Thorburn, P.J., Gaydon, D.S., Dalgliesh, N.P., Rodriguez, D., Cox, H., Chapman, S., Doherty, A., Teixeira, E., Sharp, J., Cichota, R., Vogeler, I., Li, F.Y., Wang, E., Hammer, G.L., Robertson, M.J., Dimes, J.P., Whitbread, A.M., Hunt, J., van Rees, H., McClelland, T., Carberry, P.S., Hargreaves, J.N., MacLeod, N., McDonald, C., Harsdorf, J., Wedgwood, S., Keating, B.A., 2014. APSIM - evolution towards a new generation of agricultural systems simulation. Environ. Modell. Softw. 62, 327–350. https://doi.org/10.1016/j.envsoft.2014.07.009.

Jones, J.W., Hoogenboom, G., Porter, C.H., Boote, K.J., Batchelor, W.D., Hunt, L., Wilkens, P.W., Singh, U., Gijsman, A.J., Ritchie, J.T., 2003. The DSSAT cropping system model. Eur. J. Agron. 18, 235–265.

Koo, J., 2016. DSSAT listed as one of the greatest accomplishments of the UF-IFAS. https://dssat.net/2164 (accessed November 6, 2019).

Lozza, H., 2017. Dasst: Tools for Reading, Processing and Writing dssat Files. R package version 0.3.3. Buenos Aires, Argentina. https://github.com/hlozza/Dasst.

Stanfill, B., 2015. apsimr: Edit, run and evaluate APSIM simulations easily using R. R package version 1.2. https://CRAN.R-project.org/package=apsimr.

Wickham, H., 2019. The tidy tools manifesto. https://tidyverse.tidyverse.org/articles/manifesto.html (accessed February 12, 2020).

Wickham, H., 2017. Tidyverse: Easily install and load the 'tidyverse'. R package version 1.2.1. https://CRAN.R-project.org/package=tidyverse.

Wickham, H., 2016. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag, New York.

Wickham, H., Grolemund, G., 2017. R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. O'Reilly Media, Inc.

Müller, K., Wickham, H., 2019. Tibble: Simple data frames. R package version 2.1.3. https://CRAN.R-project.org/package=tibble.

Wickham, H., Hester, J., Chang, W., 2019. devtools: Tools to make developing R packages easier. R package version 2.2.1. https://CRAN.R-project.org/package=devtools.