Jun 17th, 10:40 AM - 12:00 PM

An approach for encapsulating Fortran coded models into a R package

Carlos A. Hölbig
University of Passo Fundo, holbig@upf.br

Willingthon Pavan
University of Passo Fundo, pavan@upf.br

Jose M. C. Fernandes
Brazilian Agricultural Research Corporation, mauricio.fernande@embrapa.br

Angela Mazzonetto
University of Passo Fundo, 144981@upf.br

Tiago Zortea
University of Florida, zortea@ufl.edu

International Environmental Modelling and Software Society (iEMSs)
7th Intl. Congress on Env. Modelling and Software, San Diego, CA, USA,
Daniel P. Ames, Nigel W.T. Quinn and Andrea E. Rizzoli (Eds.)
http://www.iemss.org/society/index.php/iemss-2014-proceedings

An approach for encapsulating Fortran coded models into a R package

Carlos A. Hölbig[a], Willingthon Pavan[a], Jose M.C. Fernandes[a,b], Angela Mazzonetto[a] and Tiago Zortea[c]

[a]*Graduate Program in Applied Computing (PPGCA), University of Passo Fundo, Passo Fundo, Brazil, (holbig@upf.br, pavan@upf.br, 144981@upf.br)*

[b]*Brazilian Agricultural Research Corporation, Embrapa Trigo, Passo Fundo, Brazil, (mauricio.fernande@embrapa.br)*

[c]*Department of Agricultural and Biological Engineering, University of Florida, Gainesville, FL, USA, (zortea@ufl.edu)*

Abstract: Computer crop simulations have been successfully used around the world, specially, towards the understanding and transferring of agricultural technology. However, as soon as crop models become more complex, increases the difficulty to use, maintain, expand and improve these models. Most of crop simulation models are written in Fortran, a well-established computer language especially suited to numeric computation and scientific computing. However, Fortran has severe limitations regarding integration, interoperability and data visualization. In this study, an approach for wrapping a Fortran coded model into a R package is presented using a free and multiplatform software. By encapsulating the simulation engine in the package, one can freely manipulate both input and output data as any other regular variable eliminating the need of tabulated text files. Running the model becomes as simple as running any other R package. In addition, R can connect to databases and access data from web services. Here, two case studies are presented to illustrate the approach and to discuss the advantages on realistic crop modeling applications. These case studies include an application of the approach in a simple and in a complex crop model such as the CSM-CROPGRO-Soybean model present in the Decision Support System for Agrotechnology Transfer (DSSAT) suite.

Keywords: Crop simulation model; DSSAT; R language; RCropgro package.

1. INTRODUCTION

Crop simulation models can estimate final yield and also represent the simulation of the dynamics growth and development of crops by numerical integration (Graves et al., 2002). Dynamic process-oriented crop models predict daily growth and yield for homogenous areas. These models are site-specific, and thus need site-specific weather, soils, management, and cultivar-genetic information for most accurate predictions. These models predict the daily accumulation of biomass and its partitioning into plant components. Dynamic process-oriented crop models compute light interception, daily growth, and effects of the physical environment, including temperature, solar radiation, CO2 and soil water. Most crop models operate at the attainable yield level but ignoring pest and disease damage. The use of crop simulation models, to predict production trends and estimate risk, has become an important tool for decision making (Donatelli et al., 2002; Sinclair and Seligman, 1996). Moreover, to the scientific community such models have assisted in the organization of knowledge and hypothesis testing.

The objective of this work is to demonstrate an approach for encapsulating a crop model coded in Fortran into a R package. In this paper, we first present R and Fortran languages, followed by a broader explanation of our modular model approach. Then, the approach is demonstrated using

simple generic crop model and a widely adopted crop model known as CSM-CROPGRO-Soybean. Finally, the results are discussed and highlights and future work are presented.

2. TECHNICAL BACKGROUND

Considerable Fortran-based code has been developed over the last 40 years and it remains widely used in contemporary scientific and engineering software (Thyer, 2011). In particular, many crop models are written in Fortran. Crop models have been used as a tool by researchers, by decision-makers, by those involved in education and training and more recently for use in the study of impacts of climate variability and climate change on crop growth and development. However, as new components are added to crop growth models to expand capabilities, the models become increasingly complex (Jones et al., 2003). Thus, crop models tend be large, and using them can be time consuming with a steep learning curve.

There is a need for better data visualization that allows to explore crop models outputs in tabular and graphical formats. The solution should be as automated as possible allowing batch processing of comparisons between different scenarios and automated result analysis without human intervention between each experiment. It should require minimal reprogramming for new but similar crop models and allow new algorithms to be easily integrated with the models. Moreover, the solution should be platform-independent, free of proprietary software, based on stable technologies and compatible with models written in Fortran. The solution must also allow these models to communicate with the modern technologies for data management such as databases and web services.

R is a language and environment for statistical computing and graphics. The R base software and its contributed packages can be downloaded for free from the Comprehensive R Archive Network, available at http://cran.r-project.org/. R can be used on all modern operating systems such as Unix, Linux, Windows, and Macintosh. R also provides a well-developed programming language and a self-contained environment to perform a wide range of statistical analyzes. An R package is a default set of subdirectories and specific files compressed. This file can be generated by R, provided that files structure and folders are specified. The generated package can be distributed and easily attached to any R installation acting as a subunit of R environment. When a package is loaded, its functions become available to the user that can manipulate its inputs and outputs using all the tools provided by the language. It is also possible to load multiple R packages which increases the system robustness. For, example, packages that allow connection to databases such as `ROracle`, `RMySQL` and `RPostgreSQL` or packages that allow interaction with web services as `Rcurl, and others`. The packages are a convenient and very effective method to increase the modularity, expansibility and interoperability of the R environment. A package installation requires a single command line.

Currently, the interoperability between R and Fortran is limited to R calling Fortran code compiled as DLLs (Dynamic-link library). Indeed, many of R´s computationally intensive tasks are implemented using DLLs, typically in Fortran or C (Thyer, 2011). This strategy is convenient and efficient when the Fortran code to be executed is simple, in other words, it can be easily compiled and requires minor data passing between R and Fortran. However with the increasing model complexity the DLL approach is problematic considering the great effort needed to integrate and automatically compile the entire model.

The method proposed in this work consists primarily of creating a R package and attaching the Fortran code to it. Then functions need to be created to read the existing tabular ASCII files and convert them into R objects. This task is necessary because most current data is presented in tabular ASCII files formats. The R functions in the new package must provide a smooth translation to the present environment and backwards. The files need to be written in the same folder configured to run crop model executable. As soon as the file structure has been created, R runs the crop model executable as it is, without any change in the calling procedure, the simulator will produce the output text files. R Functions capable of reading the ASCII text files produced by the simulator are thereafter needed. These functions are important because they resume the simulation and automatically generate graphs and tables from output data.

Following the implementation of aforementioned functions on the model, there is no need to manually work with the text files or with the inner complexity of the model. Because the simulation becomes effectively a subunit of R, a single interface allows for all the needed modifications on the input data. Modular approach practices help in developing a set of simple components or modules that combined in different ways create more complex components (Bulatewicz, 2006). As the simulation becomes readily available in R environment, the integration of input and output with other packages and tools provided by R is easily accomplished. In most cases, just a few R code lines are needed. The Fortran code of the model is used without any modification; this is aligned with the principle of software reuse. This is only possible because the configuration for the simulation environment was not modified (Figure 1). Mostly, all the technologies used are free and cross-platform allowing any computer running R to execute the simulator coded in Fortran. The proposed method will be discussed in technical details in the section 4 where it will be applied on a complex, real and broadly used model called CSM-CROPGRO:Soybean.

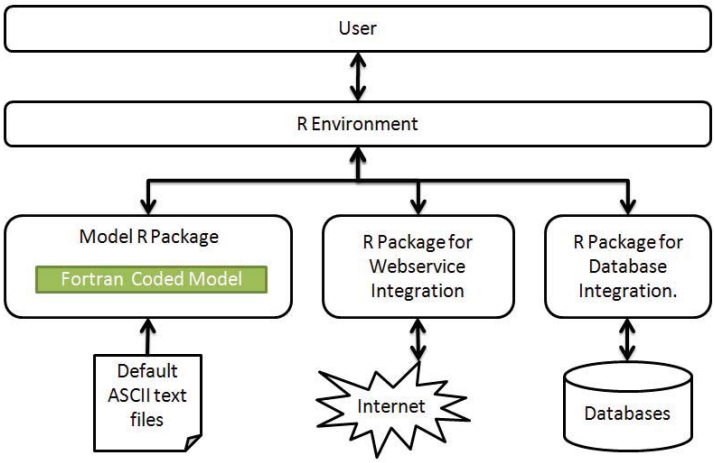

Figure 1. Diagram displaying how the legacy code interacts with R and other R packages.

3. CASE STUDY 1: RFIntegra

To better evaluate the proposed approach and to have a clear understanding of technical limitations, we first used a simple crop model. The model proposed by Porter et al. (1999) was suitable to the proposed needs because it has a simple and well documented structure. The generic crop model consists of: the main program, plant growth module, soil water balance module and a weather input routine. The modular approach described herein could be used with other computer languages as well (Porter et al., 1999).

There are three main tasks that the developed R package has to accomplish: translate the existing text files to R objects, translate R objects to text files so they can be read by the simulator and read the text files created by the simulator. The following ASCII files are required for generic crop model: `PLANT.INP`, `SOIL.INP`, `WEATHER.INP`, `IRRIG.INP`. A function in the package was created to reads each individual file. The function has a single parameter corresponding to the file path, if omitted the function will use the default file distributed within the package (Figure 2).

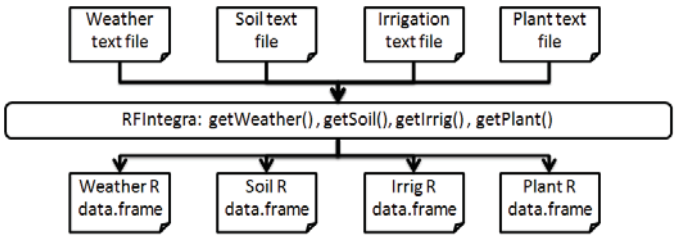

Figure 2. Structure the input text files in the simulator.

```
# Using functions provided by the RFIntegra package to
# convert text files into R objects.
> weather<-getWeatherTxt("//RFIntegra//data//WEATHER.INP")
> tail(weather)
DOY solar rad(MJ/m2) tmpmax(c)      tmpmin(c)    rain(mm)     par(E/m2)
725 86360 6.1           15.6        7.2          0.0          12.8
726 86361 6.1           16.7        8.3          0.0          12.8
727 86362 12.3          18.3        8.9          0.0          25.9
728 86363 10.8          20.0        6.1          0.0          22.7
729 86364 9.2           18.9        7.2          0.0          19.4
730 86365 1.3           15.6        8.9          8.9          2.7
```

The R commands shown above read the `WEATHER.INP` and make its content available on the `weather` object, that is a data frame already formatted to the `runSimulation` command. Once input objects are available in R, it is possible to call the core function of the package that will, in fact, execute the simulator. It requires one data frame for each input file and simulation parameters, in this example bellow the sowing date, using the command "`runSimulation(weather,plant,soil,irrig,params)`".

```
# Running the fortran simulation using objects
# created by the get text functions.
> params<-list(doyp=121)
> runSimulation(weather,plant,soil,irrig,params)
```

When the `runSimulation` function is called, the package will read the data frames provided and write on the disk all the ASCII files needed by the simulator. After the package will run the simulator by the same manner it was done traditionally but using the produced text files. Following a successfully `runSimulation` command, the results are written in the disk as ASCII files, the package provide functions to read these files and convert them into data frames (Figure 3). The resulting data frames can be handled in R as any other R object, the `runSimulation` command can be called again with different parameters representing experiments..

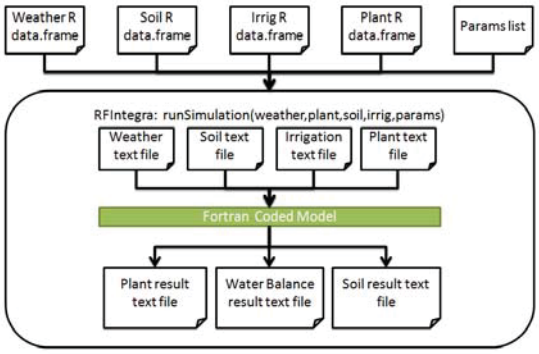

Figure 3. Interaction of R objects with the simulator coded in Fortran.

The core feature of the proposed approach is successfully "hiding" all the complexity associated to the execution of a legacy Fortran model behind a R package. The R package can be freely distributed and run on any computer with R installed. This improvement had been achieved without a single modification on the Fortran code, thus no further crop model validation is needed.

The package provides a convenient interface for handling both input and output facilitating in crop model applications. In addition, it would be possible comparing automatic batch for different scenarios. It also allows the simulation to take advantage of all the tools available in the R language, from database access to graphical displays and statistics functions. Writing model output in the disk slowed the process. However, the increase in processing time was almost unnoticeable. This is an important conclusion because it could allow thousands of scenarios comparisons using the looping

structures of R. Moreover, since R version 2.13, it is possible to use just in time compilation (e.g. `compiler` package) to improve the runtime performance of computer programs.. Along with the combination of C/C++ code with R code (for example, through the `Rcpp` and `inline` packages), the use of these tools can provide a gain in performance of programs running on R.

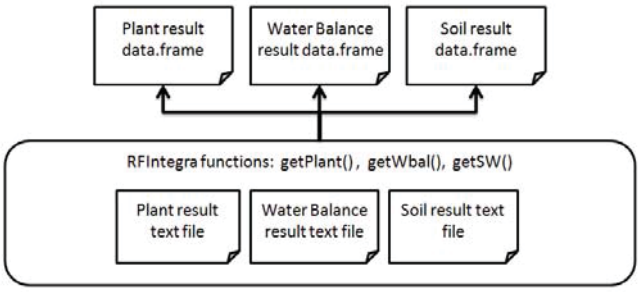

Figure 4. Interaction of text files produced by the simulation with the R.

4. CASE STUDY 2: RCropgro PACKAGE

The DSSAT is a collection of independent programs that operate together; crop simulation models are at its center. Databases describe weather, soil, experiment conditions and measurements, and genotype information for applying the models to different situations (Jones et al., 2003). The DSSAT is composed by numerous crop models designed for global application independent of location, season or cultivar (Verhagen et al., 2001). It also can combine the data from soil, weather and crop to simulate the results of many years of crop management. CSM-CROPGRO-Soybean is one of the Fortran models that composes the DSSAT suite. The CSM-CROPGRO-Soybean has been calibrated and used in Brazil (Banterng et al., 2009).

The first step to apply the approach in a crop simulation model is to map its inputs and outputs. The CSM-CROPGRO-Soybean model uses the following ASCII files: `PLANT.INP`, `SOIL.INP`, `WEATHER.INP`, `IRRIG.INP` and `SIMCTRL.INP`. Each of these files will need a R function in the package capable to read and translate them into R objects. The user should pass to the function a parameter informing the path of the text file containing the desired inputs, if omitted the default file distributed with the package will be used. In the configuration a data folder is specified to hold weather, soil, plant and planting details files. If the path is omitted, the function `getPlant()` has a default template.

The CSM-CROPGRO-Soybean model compilation needs the IFORT compiler, which is proprietary and needs advanced computer knowledge to be successfully installed. Besides that, CSM-CROPGRO-Soybean model does not have an automatic compilation file (`Makefile`) and building one is not always a trivial task. In this case, the best approach is to manually compile the model and have different package for each operational system, the binary files must be placed in the `inst` folder to get copied to the package, in the `root` folder, during installation. The package loading is the process whereby all the functions become available to the R environment. Furthermore all the dependencies of the package are loaded and commands readily available. The `RCropgro` package depends on the package `Gdata`. This R package improves the R capacity to handle tabular ASCII text files and Excel sheets and it is available at http://cran.r-project.org/web/packages/gdata. Furthermore during the loading process the files that were not integrated are copied to the simulation folder, there files were not integrated because they are not modified so frequently, but it is an opportunity for future work. The files are: Soil configuration `EB.SOL`, `SOIL.SOL`, `SOILN040.SOL`, `SOMFIX_C.SOL` e `SOMFRACTIONS_C.SOL`, cultivar property `SBGRO040.CUL`, ecotype `SBGRO040.ECO`, pests `SBGRO040.PST` and species configuration `SBGRO040.SPE`. Although these files are not integrated, they can be modified by any text editor, the folder were they are located is shown by the welcome message when the package is loaded. Detailed information about Cropgro configurations can be obtained in the Hunt et al. (2006). The execution of the CROPGRO-Soybean model is performed by the `runSimulation` command. This command needs both `weather` and `SBX` (Experiment Input File) variables to be informed by parameter plus the experiment number.

During this process, as detailed in Figure 4, these R variables will be converted again in text files and written in the simulation folder.

Once the simulation is executed, output files are written in text files. The package verifies if the output files have been successfully generated. The results can retrieve using the `getResult` function. The CSM-CROPGRO-Soybean model generates many output text files detailing each daily step. For simplification, only the summarized files were integrated. Results can be retrieved by the functions `getOvrResult` and `getSumResult` witch converts the text files `OVERVIEW.OUT` and `Summary.OUT` respectively into structured R data frames.

5. DISCUSSION

The proposed approach not only greatly simplified the execution CSM-CROPGRO-Soybean model but also allowed that the inputs and outputs to be manipulated directly on R. As long the input data is translated, the simulation can be executed repeatedly by varying parameters to compare different scenarios. As a proof of concept, the RCropGro package was used to run the CSM-CROPGRO-Soybean model aiming to represent a soybean experiment with 28 treatments. The experiment aimed to compare 14 sowing dates: October (1,8,15,22,29), November (5,12,19,26), December (3,10,17,24,31) with 2 different cultivars: Bragg and Don Mario. Phenological data were compared: Anthesis day (dap), Pod 1 day (dap), Full pod day (dap) and Physiological maturity day (dap), all measured in days after plantion. This example, aims to demonstrate the potential that the package gives to legacy simulations (see Figures 5 and 6). With just few modifications on this code it would be possible to plot any simulated variable.

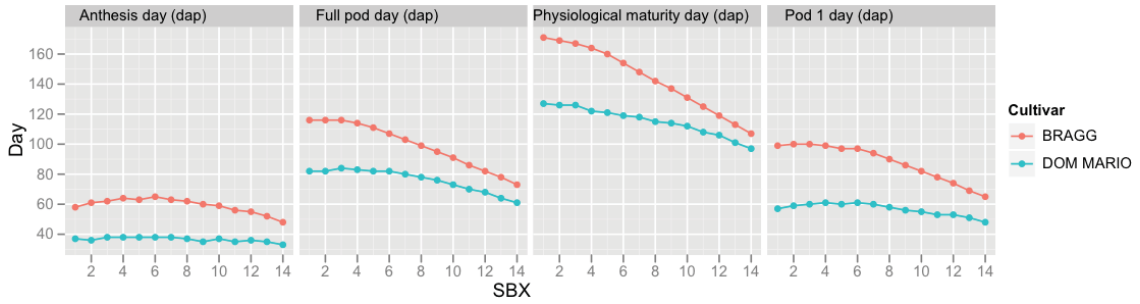

Figure 5. Simulated phenology of two soybean cultivars using CROPGRO-Soybean model directly executed in R.

6. FINAL REMARKS AND FUTURE WORK

The `RCropgro` package attained the objectives that have been proposed. It allowed the revitalization of highly mature legacy of crop models coded in Fortran under a R library effectively reducing most of the problems bound to the legacy code. The approach was carried out directly in R using its looping functions and `RCropgro` package to explore different scenarios. Furthermore, the `RMySQL` package allowed data to be retrieved from MySQL databases and used directly in the CSM-CROPGRO-Soybean model. There was no need to modify the Fortran code. The approach uses only free multi-platform technologies. Looking ahead, the authors envisage the crop model community being able to create functions in R and distribute them as open-source as well. Other R packages with specific crop model functions can be created having `RCropgro` as prerequisite, this would allow all the scientific community to take advantage of them.

As the proposed approach has been applied to the model Cropgro-Soybean model, it could be applied to any legacy coded model dealing with ASCII text files. For instance it could be applied to other models of the DSSAT suite. The task would not require deep knowledge about crop models.

Another important contribution would be to develop a framework in R that would allow access and modification in the simulation. With this framework it would be possible to change the internal

behavior the simulation just by changing data sources and setting values in R, eliminating the need of knowledge in FORTRAN. Several models have been integrated with mapping APIs to generate maps to gain a broader view of the agronomic scenario. This approach allows the simulation to be run thousands of times generating maps comparing on different regions or impact of possible climate change on yield, for example. Furthermore, as more information is available, researchers will find more and more ways to use it needing faster results. As each experiment is independent, they could be run different parallel processes or threads nearly multiplying the speed of the multiple simulations by the number of allowed threads. That would allow thousands of experiments to be run in high performance environments.

```
1. #Example using the experiments already configured on the package.
2. library(RCropgro)
3. library(ggplot2)
4. #### Running the first 14 experiments configured in the SBX file.
5. #### These experiments correspond to the Bragg cultivar.
6. RCWeather<-getWeatherTxt()
7. RCSBX<-getSBXTxt()
8. n<-c(1:14)
9. count=1
10.   for(i in n){
11.   runSimulation(experiment=i,RCSBX,RCWeather)
12.   overall<-getOvrResult()
13.   if(count==1)comp<-overall$growthAndDev$Simulated
14.   else comp <-cbind(comp,overall$growthAndDev$Simulated)
15.   cat("Experiment",count,"of",length(n),".")
16.   count=count+1
17.   }
18.   dimnames(comp)<-list(overall$growthAndDev$Variable,c(1:14))
19.   compBragg<-comp
20.   #### Running 14 experiments of the Don Mario cultivar.
21.   n<-c(15:28)
22.   count=1
23.   for(i in n){
24.   runSimulation(experiment=i,RCSBX,RCWeather)
25.   overall<-getOvrResult()
26.   if(count==1)comp<-overall$growthAndDev$Simulated
27.   else comp <-cbind(comp,overall$growthAndDev$Simulated)
28.   cat("Experiment",count,"of",length(n),".")
29.   count=count+1
30.   }
31.   dimnames(comp)<-list(overall$growthAndDev$Variable, c(1:14))
32.   compDom <- comp
33.   ##### Prepare data for the Chart
34.   molten <- melt(compBragg[c(1:4),]) ## Selecting only 4 variables
35.   molten <- cbind(molten, Cultivar='BRAGG')
36.   molten2 <- melt(compDom[c(1:4),])
37.   molten2 <- cbind(molten2, Cultivar='DOM MARIO')
38.   molten<-rbind(molten,molten2)
39.   dimnames(molten)<-list(dimnames(molten)[[1]],c("X1","SBX", (+)
40.   "Day","Cultivar"))
41.   ##### Generate Chart
42.   chart <- ggplot(molten, aes(SBX,Day, colour=Cultivar)) + geom_point()
43.   chart <- chart+ geom_line()
44.   chart + facet_grid( . ~ X1 )
```

Figure 6. `RCropgro` package - code example.

7. REFERENCES

Banterng, P., Hoogenboom, G., Patanothai, A., Singh, P., Wani, S. P., Pathak, P., Tongpoonpol, S., Atichart, S., Srihaban, P., Buranaviriyakul, S., Jintrawet, A., and Nguyen, T. C. (2009). Application of the Cropping System Model (CSM) - CROPGRO- Soybean for Determining Optimum Management Strategies for Soybean in Tropical Environments. Journal of Agronomy and Crop Science 196(3):231-242. DOI: http://dx.doi.org/10.1111/j.1439-037X.2009.00408.x

Bulatewicz, T. (2006). Support for Model Coupling: an interface-based approach. PhD thesis, Department of Computer and Information Science and the Graduate School, University of Oregon.

Donatelli, M., Van Ittersum, M.K., Bindi, M., and Porter, J.R. (2002). Modelling cropping systems - highlights of the symposium and preface to the special issues. European Journal of Agronomy, v.18, Issues 1-2, p.1-11. DOI: http://dx.doi.org/10.1016/S1161-0301(02)00104-1.

Graves, A.R., Hess, T., Matthews, R.B., Stephens, W., and Middleton, T. (2002). Crop simulation models as tools in computer laboratory and classroom based education. Journal of Natural Resources and Life Science Education 31:48-54.

Hunt, L., Jones, J., Hoogenboom, G., and White, J. W. (2006). Icasa version 1.0 data standards for agricultural research and decision support. Technical report 01/0264, International Consortium for Agricultural System Applications.

Jones, J., Hoogenboomb, G., Porter, C., Boote, K., Batchelor, W., Hunt, L., and Wilkens, P. (2003). The dssat cropping system model. European Journal of Agronomy, 18:235–265.

Pavan, W. (2007). Técnicas de Engenharia de Software Aplicadas à Modelagem e Simulação de Doenças de Plantas. PhD thesis, Universidade de Passo Fundo.

Porter, C. H., Jones, J. W., and Braga, R. P. (1999). An approach for modular crop model development. Research report 99-0701, Agricultural and Biological Engineering Department.

Sinclair, T.R., and N.G. Seligman. (1996). Crop modelling: From infancy to maturity. Agron. J., 88:698-703.

Thyer, M. (2011). The open source rfortran library for accessing r from fortran, with applications in environmental modelling. Environmental Modelling & Software, 26:219–234.

Verhagen, A., Conijn, J., and Schapendonk, A. (2001). Quickscan of simulations models. Scientific report 130, Wageningen: Plant Research International B.V.