Fusing R and BUGS through Wine

An Introduction to Package `rbugs`

by Jun Yan

Historical Review

BUGS (Spiegelhalter et al., 1996) has been a very successful statistical software project. It is widely used by researchers in many disciplines as a convenient tool for doing Bayesian statistical analysis. The BUGS developers have shifted developmental efforts to WinBUGS, and the classic BUGS is not being developed further. The current release WinBUGS 1.4 comes with a scripting facility which permits batch running, and also therefore the ability to run WinBUGS from other programs.

Fusing R and BUGS has been available from several sources. On Windows systems, it dates back to Kennith Rice's package `EmBedBUGS` (Rice, 2002). `EmBedBUGS` is available in an Windows self-extracting archive and is not in the standard format of an R package. First version adapted from `EmBedBUGS`, Andrew Gelman's collection of functions `bugs.R` (Gelman, 2004) has evolved into a comprehensive tool, which can run WinBUGS 1.4 from R, collect the MCMC samples, and perform basic output analysis. This collection was recently packaged by Sturtz and Ligges (2004) as `R2WinBUGS`, with some tools (including print and plot methods) for output analysis slightly changed from Andrew Gelman's original function.

On Linux/Unix systems, however, less work has appeared since the pause of the support for classic BUGS. Plummer (2003) reported experience and some known problems of running WinBUGS under Wine, "an Open Source implementation of the Windows API on top of X and Unix" (Wine, 2004). With Wine, it is possible to provide a facility of fusing R and BUGS on a Linux system similar to what's available on a Windows system. This is what package `rbugs` aims at.

Design

The powerfulness of BUGS (In the sequel, I use BUGS instead of WinBUGS when there is no confusion in the context, hoping some day classic BUGS will be supported.) lies in that, with a straightforward syntax for model specification, it provides a universal MCMC sampler of posterior distributions for rather complicated problems. Users do not need to worry about how the MCMC samples are actually drawn. The design philosophy of `rbugs`, therefore, is to take the advantage of the universal MCMC sampler of BUGS through an interface as simple as possible, and return the MCMC samples in a format which can be fed into other R packages specializing in Bayesian output analysis, such as `boa` (Smith, 2004) and `coda` (Plummer et al., 1996). In addition, users (particularly those who are uncomfortable with point-and-click) enjoy accesses to various files generated during the preparation of running BUGS in batch-mode.

Compared to package `R2WinBUGS`, `rbugs` is different in the following sense: 1) It does not provide Bayesian output analysis and only serves as a fuse to connect R and BUGS; 2) It provides access to automating the preparation of script file, model file, data file, and initial value file, which are needed by running BUGS in batch-mode; and 3) Its main target users are Linux users having access to Wine.

Configuration

Package `rbugs` has been tested on both Linux and Windows. After installation, it's worth setting two environment variables in the '.Renviron' file to save some typing: `BUGS` and `WINE`. These two variables store the full name of the executables of BUGS and Wine, respectively. They are used as the default values in function `rbugs`. The following is an example on my machine:

```
BUGS="c:/program files/winbugs14/winbugs14.exe"
WINE="/var/scratch/jyan/wine-20040408/wine"
```

The definition of `WINE` is only necessary if BUGS is to be used via Wine. In that case, the wine configuration in './wine/config' in the home directory will be processed by an internal function to create a map from the Windows drives to the native directories. Further discussion about the usage via wine is presented next.

Run BUGS in a Single Call

To run BUGS in batch-mode, a minimum of four files are needed as input: a script file, a model file, a data file, and an initial value file for each chain to be run. Except the model file, other three types of files can be generated. The model file would need to be written by a user outside of R. The output from BUGS are saved in files specified in the script file, and can be read into R and used for convergence and output analysis. From sufficient information collected from its arguments, such as the data list, parameters to be monitored, number of chains, etc., function `rbugs` generates the data file, initial value files, and script file that are needed, calls BUGS through an OS-specific system call, and returns the MCMC output as a list of matrix. The returned object can be

further processed by packages boa and coda, taking the advantage of various native analysis available in R. An example is provided with the pumps data in the Example Volume I of BUGS:

```
> ? pumps
```

Experience with Wine

On a RedHat 3.0 workstation, I experimented running WinBUGS via Wine 20040408. As reported by Plummer (2003), buttons still don't respond to clicks. One would have to use the return key after pointing to a button. This may not be a problem for people who do not like using a mouse anyway. Fortunately, the batch-mode works fine and the results from some examples I tried are the same as those obtained from a Windows system.

Installation guide for Wine can be found from its website. For people who don't have root access, it's sufficient to just compile it and set WINE as the full name of the Wine executable in the compiling directory. The compiling is straightforward.

When using rbugs, one needs to pay attention to the difference in two of its arguments: workingDir and bugsWorkingDir. On a Windows system, they should be the same. But on a Linux system, workingDir refers to the directory that's recognizable by native operations, while bugsWorkingDir refers to the same directory as workingDir but translated to a Windows directory recognizable by WinBUGS via Wine. For example, on my system, drive C is defined in the Wine configuration:

```
[Drive C]
"Path" = "/var/scratch/jyan/c"
```

If I would like to use

```
bugsWorkingDir="c:/tmp",
```

then I would need to have

```
workingDir="/var/scratch/jyan/c/tmp".
```

With these straightened out, the pumps example can be run on a Linux system. Same as on a Windows system, rbugs will launch BUGS. In the current release of rbugs, the debug information from Wine is redirected to a temporary file and deleted on exit.

The configuration information of wine is usually stored in '.wine/config' in the home directory. An internal function processes this config file and stores the drive mapping between Windows and the native Linux system in a internal data frame .DriveTable. When using rbugs, if workingDir is the default, NULL, then bugsWorkingDir is translated using the drive mapping table.

Preparing Files for BUGS Batch-mode

Often times, one would like to use a call of rbugs to do some exploration, checking if the model and the data compile fine in BUGS. It would be useful to have the generated script file, data file, and initial value files available for using BUGS directly in other circumstances. There is no difficulty in generating the script files. But for the data file and the initial value files, the format of the data becomes an important issue. In the WinBUGS 1.4 manual, under the section of Model Specification, formatting of data is discussed. It reads that BUGS can take read files created from the S-Plus dput function. Unfortunately, this is not (or no longer) true for both the most recent versions of S-Plus and R. Let's look at R-1.9.0 only:

```
> a <- matrix(c(314159265358979, 0.0001,
                -0.0001,         0.05), 2, 2)
> dput(list(a = a), "tmp")
> file.show("tmp")
structure(list(a = structure(c(314159265358979,
1e-04, -1e-04, 0.05), .Dim =
as.integer(c(2, 2)))), .Names = "a")
```

A BUGS user immediately sees that BUGS will complain when it reads this! Besides the extra characters of "as.integer" and ".Names", there are less documented subtle issues: 1) "e" should be "E"; 2) "1e" should be "1.0E"; and 3) the first number exceeded 14 digits.

In not necessarily the most efficient way, the function format4Bugs converts the data to characters with formatC and then uses a modification of a format data function by Kennith Rice to return hopefully the right format for BUGS.

Using format4Bugs, functions genDataFile and genInitsFile prepare data file and initial value files. Function genBugsScript generate a script file. All these files are accessible by users and hence ease the usage of BUGS in other circumstances.

Remarks

Since Wine is built upon X windows, WinBUGS would not run from an ssh terminal without X windows support. Many people had wished the classic BUGS were supported. That not happening soon, it would be desirable to have a better supported command line interface of WinBUGS, so that launching the GUI becomes an options.

As an infrequent Windows user, I am more oriented to experimenting and supporting fusing R and BUGS through Wine on Linux systems. Windows users are referred to package R2WinBUGS. A forthcoming paper by the package authors will provide detailed demonstration and become a standard reference.

Plummer (2004) just released 0.50 of JAGS. Quote from Martyn Plummer: "JAGS is Just Another Gibbs Sampler - an alternative engine for the BUGS language that aims for the same functionality as classic BUGS. JAGS is written in C++ and licensed under the GNU GPL. It was developed on Linux and also runs on Windows." The functions in package `rbugs` can also be used to prepare files for JAGS. I am looking forward to seeing the growth of JAGS.

I also tried using R for Windows through Wine. It worked last winter with Wine 20031016, but is not working with Wine 20040408 now. Unfortunately, since my Wine 20040408 was compiled after my system has been recently upgraded to Red Hat Workstation 3.0, I cannot tell which change has caused it.

Bibliography

Gelman, A. (2004), "bugs.R: functions for running WinBugs from R," http://www.stat.columbia.edu/~gelman/bugsR/. 19

Plummer, M. (2003), "Using WinBUGS under Wine," http://calvin.iarc.fr/bugs/wine/. 19, 20

Plummer, M. (2004), "JAGS version 0.50 manual," http://www-fis.iarc.fr/~martyn/software/jags/. 20

Plummer, M., Best, N., Cowles, K., and Vines, K. (1996), "coda: Output analysis and diagnostics for MCMC," http://www-fis.iarc.fr/coda/. 19

Rice, K. (2002), "EmBedBUGS: An R package and S library," http://www.mrc-bsu.cam.ac.uk/personal/ken/embed.html. 19

Smith, B. (2004), "boa: Bayesian Output Analysis Program for MCMC," http://www.public-health.uiowa.edu/boa. 19

Spiegelhalter, D. J., Thomas, A., Best, N. G., and Gilks, W. (1996), *BUGS: Bayesian inference Using Gibbs Sampling, Version 0.5, (version ii)* http://www.mrc-bsu.cam.ac.uk/bugs. 19

Sturtz, S. and Ligges, U. (2004), "R2WinBUGS: Running WinBUGS from R," http://cran.r-project.org/src/contrib/Descriptions/R2WinBUGS.html. 19

Wine (2004), "Wine," http://www.winehq.org. 19

Jun Yan
University of Iowa, U.S.A.
jyan@stat.uiowa.edu

R Package Maintenance

Paul Gilbert

Introduction

Quality control (QC) for R packages was the feature that finally convinced me to maintain R packages and also run them in S, rather than the reverse. A good QC system is essential in order to contain the time demands of maintaining many packages with interdependencies. It is necessary to have quick, easy, reliable ways to catch problems. This article explains how to use the R package QC features (in the "tools" package by Kurt Hornik and Friedrich Leisch) for ongoing maintenance and development, not just as a final check before submitting a package to CRAN. This should be of interest to individuals or organizations that maintain a fairly large code base, for their own use or the use of others.

The main QC features for an R package check that:

- code in package directory R/ is syntactically correct
- code in package directory tests/ runs and does not crash or stop()
- documentation is complete and accurate in several respects
- examples in the documentation actually run
- code in package directory demo/ runs
- vignettes in package directory inst/doc/ run

These provide several important features for package maintenance. Developers like to improve code, but documentation updates are often neglected. A simple method to identify necessary documentation changes means documentation maintenance is (almost) painless. The QC tools can be used to help flag when documentation changes are necessary. They also ensure that packaged code can be quickly tested to ensure it works with a new version of R (or a new compiler, or a new operating system, or a new computer). The system explained below also helps check dependencies among functions in different packages, easing development by quickly identifying changes that break code in other packages.