User's Guide to the R Package PBSadmbwin[1]

by Jon Schnute and Rowan Haigh
Pacific Biological Station, Nanaimo, BC, Canada

What is PBSadmb?

PBSadmb provides an R interface for developing models with the open version of a software package called AD Model Builder (ADMB). An R function handles every command normally associated with ADMB, including all steps required to make executable files. Consequently, the package enables an R user to handle all aspects of ADMB development with R script files. It includes a facility for ensuring that variable names match between the source file for ADMB (called a *template*) and a corresponding source file for R.

PBSadmb depends heavily on another R package PBSmodelling. We use this to implement a Graphical User Interface (GUI) that greatly facilitates ADMB model development. A user can edit code, test it rapidly, and inspect the results of analyses, including Markov chain Monte Carlo (MCMC) simulations. A new user can learn key aspects of ADMB template code with the examples provided.

In short, PBSadmb extends R to provide full support for ADMB. R users (affectionately called useRs) can regard ADMB as just another R application. Currently we are developing separate PBSadmb packages for Windows (PBSadmbwin), Linux (PBSadmblin32 and PBSadmblin64), and Mac OS (PBSadmbmac). However, since R supports many common operating systems, we anticipate one R package that will eventually provide a uniform interface across all supported platforms.

You can obtain the PBSadmb packages from the web site: http://code.google.com/p/pbs-admb/.

What is PBS?

The initials 'PBS' refer to the Pacific Biological Station, a major fisheries laboratory operated by Fisheries and Oceans Canada on the Pacific coast in Nanaimo, British Columbia, Canada. For more information, see: http://www.pac.dfo-mpo.gc.ca/sci/pbs/.

We have developed a number of packages for R, each starting with the acronym PBS. Three of these (PBSmapping, PBSmodelling, and PBSddesolve) are available on the Comprehensive R Archive Network (CRAN, http://cran.r-project.org/). Because PBSadmb currently includes binary libraries and executable code, we are distributing it from the Google Code site mentioned above rather than from CRAN.

[1] PBSadmbwin is the name of the PBSadmb package for the Windows operating system only. Throughout this document, the term PBSadmb refers to this specific Windows version.

What is ADMB?

The remarkable software package ADMB offers powerful tools for estimating parameters and their uncertainty from complex statistical models. It uses automatic differentiation (sometimes called algorithmic differentiation) to compute function gradients needed for efficient estimation. It includes robust algorithms for modal estimation and MCMC sampling from Bayesian posterior distributions. Other common inference methods, such as likelihood profiles are also supported. ADMB allows you to examine your data with any statistical model that has a properly defined likelihood function or Bayesian posterior. The model can have hundreds or even thousands of unknown parameters that require estimation.

Originally, ADMB was developed commercially by its principal author David Fournier and the company Otter Research Ltd. (http://www.otter-rsch.com/). It quickly gained wide use in fishery data analyses, although it has potential value in many scientific fields. In 2008, the ADMB Project (http://admb-project.org/) acquired rights to the software and placed it in the public domain. A number of people worked hard to make this possible, and we thank all of them for their efforts. In particular, John Sibert plays a key role in developing and maintaining the Project web site.

How do ADMB and R tie together?

As useRs know, the R software environment easily accommodates external programs. R packages (such as `PBSddesolve`) often include C/C++ code directly, and the packaging system automatically compiles the code for all supported operating systems. ADMB is a bit unusual in this context because it necessarily involves a C++ environment that cannot be entirely masked by R. The automatic differentiation algorithms, implemented with C++ classes, require a user to express the posterior or likelihood in C++. Fournier had the ingenious idea of making this process as easy as possible with a *template* that handles most of the annoying bookkeeping, so that a user need only write code (very similar to R code) that expresses the model analytically. Program development involves three distinct steps: (1) converting the template to true C++ code, (2) compiling the C++ code, and (3) linking the resulting object module to ADMB libraries. The complete cycle makes an executable file that recognizes a variety of command line arguments.

`PBSadmb` implements these three steps with the R commands `convAD` (convert to C++), `compAD` (compile C++), and `linkAD` (link to libraries). A composite command `makeAD` performs all three steps sequentially. The ADMB libraries come as part of the `PBSadmb` installation, so the required paths to them are automatic. However, a user *must* declare a path to the relevant C++ compiler and the text editor used to develop source code. A user can also override the default ADMB path within `PBSadmb`, but we recommend against it. This version of `PBSadmb` is designed to work explicitly with the GNU C compiler used when building R packages.

The most convenient interface between ADMB and R comes from the GUI shown below in Figure 1. This allows a user to explore all aspects of ADMB model development. The interface emphasizes four distinct phases:

- **Initialize** the package with appropriate paths , check that they make sense, and save them in a file normally called `Adopts.txt`.
- **Make** the executable file for a chosen prefix, with options between "Safe" and "Optimized" compilation and a choice to have random effects or not.
- **Run** the executable code with suitable command line arguments, where the "All args" button shows all available arguments. The interface gives particular support for generating MCMC samples and likelihood profiles. The "Custom" button supports arbitrary "AD args".
- Inspect the **Output** by "View"ing various reports or "Import"ing them into the R working environment. Again, we give special support to MCMC samples with plots that allow a user to inspect the sampled chain. A user can choose variables for plotting and thin the current chain.

Experienced ADMB users know that ADMB leaves many "footprints", i.e., files in the current working directory. The interface gives you "Clean" buttons to help clean them up. To make things easy, each "Clean" button activates a second GUI that displays potential files associated with the project prefix, as well as other debris files spawned by ADMB. The user can fine-tune the selection using the "Select" and "Deselect" buttons. When the "Clean" button is pressed, a final prompting GUI pops up to confirm deletion of the selected files. Once the files have been deleted, the Clean window remains and the user can choose another prefix (manually typing or pressing the selection button ">") AND hitting the "Refresh" button. This causes the GUI to rebuild itself with files having the newly selected prefix. If no additional files are apparent, the Clean window disappears. Files with suffixes `.tpl`, `.dat`, `.pin`, `.r`, and `.pdf` are never picked for potential deletion. Be careful when cleaning; for example don't delete an output file until you're sure you're ready to do so.

We encourage you to experiment with the GUI. You can quickly see the functionality available in the main menu items. <Edit> allows you to edit the main project files, and <View> displays the output files. <Examples> copies various examples (discussed below) into your working directory. <Package> shows the R code for this package and the Window description file used to create the GUI in Figure 1. <Help> points to manuals in the package, online resources, and this User's Guide.

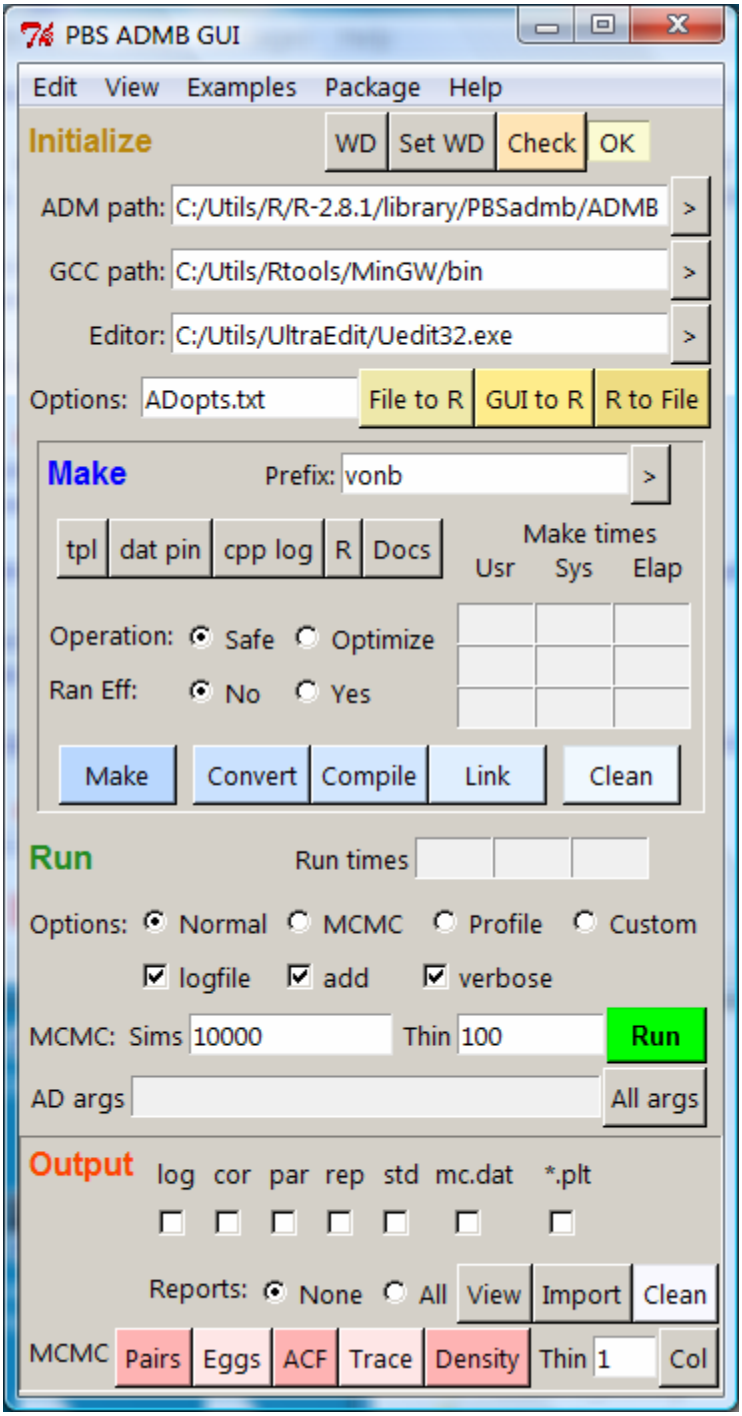

Figure 1. The graphical user interface (GUI) in `PBSadmb`, generated by the R command `admb()`.

Buttons ">" in the "Initialize" and "Make" sections allow a user to browse for available choices. Text boxes in the "Make" section show the times required for converting (row 1) compiling (row 2), and linking (row 3). The R function `proc.time` reports the 'user time' and 'system time', as well as the elapsed time, and these correspond to the three columns in the interface. Similarly, text boxes in the "Run" section show the run times.

How do I install PBSadmb?

Installation is easy, as it is for most R packages, although this one has a few extra twists. Essentially, you need to install R, PBSadmb, the R toolkit required for package development, and a text editor suitable for writing templates and viewing reports. Then you need to run R, load PBSadmb, and give it some configuration information. At this point, you have a working version of the interface in Figure 1. Here are the details.

Step 1. Install the current Windows version of R from the web site http://cran.r-project.org/. We assume that you have enough familiarity with R to do this without difficulty. In this example, we also assume that you've used the directory C:\Utils\R\Rx.xx, where x.xx is the current version number of R.

Step 2. Run R and install the current version of the package PBSmodelling (<Packages>, <Install package(s)...> in the R GUI). Enter the command require(PBSmodelling) and check that the R console reports version 2.01 or later.

Step 3. Go to the web site http://www.murdoch-sutherland.com/Rtools/, and download the file "Rtools28.exe". Run this executable file, and install the R tools in a directory of your choice. In this example, we assume you've used the directory C:\Utils\Rtools. Take a moment to inspect the installed files. You should find a subdirectory C:\Utils\Rtools\MinGW\bin that contains the GNU compilers, including g++.exe. If you type

```
C:\Utils\Rtools\MinGW\bin\g++ --version
```

in a command window, you should see the result

```
g++ (GCC) 4.2.1-sjlj (mingw32-2)
```

You're using version 4.2.1, where the sjlj refers to "Short Jump/Long Jump". You also have all the tools required to build R packages like this one.

Step 4. Obtain a good text editor that you can use for code development. The Windows Notepad will work, but much better options are available. We happen to use a commercial program called UltraEdit (http://www.ultraedit.com/), but you may prefer to get something free. Our editor supports syntax highlighting and displays multiple files in a single window, with tabs to select among them.

Step 5. Go to the web site http://code.google.com/p/pbs-software/, and download PBSadmb_y.yy.zip, where y.yy is the most recent version available. Then install PBSadmb from this zip file. (In the R GUI, click <Packages> and select "Install package(s) from local zip files".)

Step 6. Run R in an empty working directory. Then type these two commands into the R console:

```
> require(PBSadmb)
> admb()
```

The GUI should appear, along with a warning message that you have no AD options file. You can use the GUI to set the necessary paths. The "ADM path" should be OK, consistent with the directory you chose in Step 1. On the "GCC path" line, click ">". This brings you into an interface where you can locate the bin directory (with g++.exe) selected in Step 3. Similarly,

click ">" on the "Editor" line to select the executable file for the editor in Step 4. (By default, the GUI points to the Windows Notepad, but hopefully you have a better choice.)

Step 7. Next click "GUI to R" to create an R variable `.Adopts` that contains your specified options. As usual, you can inspect it in the R console by typing its name. Next click "R to File". This creates a file `Adopts.txt` in your current working directory that you can inspect with the text editor. Finally, click the "Check" button. If everything is OK, you should see "OK" in the adjacent text box. The message "Fix" means that something critical can't be found on the paths you've specified. Either you haven't installed something correctly, or one of the paths is wrong.

In the future, when you issue the R command `admb()` with this working directory, the file `Adopts.txt` will automatically determine the paths in the GUI. Furthermore, you can copy this file to any other directory from which you want to use `PBSadmb`. Conceivably, you might use different option files for projects in different directories

How can I see ADMB in action?

`PBSadmb` includes a number of examples that teach new users (and remind experienced users) how to write, test, and implement an ADMB template. To see them click <Examples> on the interface menu. If you click one of them (the file prefix), the program will load all related files into the current working directory. Typically, these have the suffixes

- `.tpl` – the ADMB template file;
- `.dat` – the data used for this template;
- `.pin` – initial values for the parameter estimates;
- `.r` – R code that can be sourced to obtain an extended analysis using both ADMB and R;
- `.pdf` – documentation for this example.

We encourage new users to examine the files in the following order:

`Simple`, adapted from an example in the ADMB manual, codes the likelihood for regressing a vector `y` on a vector `x`. Take special note of how code is written for the four SECTIONs (DATA, PARAMETER, PROCEDURE, REPORT). Values initialized in the DATA_SECTION come from `simple.dat`, and values initialized in the PARAMETER_SECTION come from `simple.pin`.

`SimpleMC`, a variant of `simple`, can give a Bayesian posterior sample of the parameters. The GUI allows you to perform and "MCMC" run. You can then view results visually with plots generated from the "Output" section of the GUI.

`SimplePBS`, a variant of `simpleMC`, has a REPORT_SECTION written explicitly for `PBSadmb` to ensure that variable names in R code match those from ADMB. In this case, the file `simplePBS.r` (1) makes `simplePBS.exe` from `simplePBS.tpl`, (2) runs `simplePBS.exe`, (3) loads the data from `simplePBS.rep` into R, preserving variable names, and (4) produces a standard regression plot for the data exported imported from `simplePBS.rep`.

vonb, similar to `simplePBS`, estimates parameters for a Bertalanffy growth curve. It can also generate a likelihood profile for the parameter `Linf`, renamed for this purpose as `VonBLinf`. In this case, ADMB generates a file named `VonBLinf.plt`, with the parameter name prefix, *not* the prefix `vonb`.

catage, taken from ADMB web sites, implements a more complex model designed for estimating biological parameters from fishery data on catch and age structure. In the case, the code allows a user to compute a likelihood profile for the predicted biomass `pred_B`.

pheno, also taken from ADMB web sites, implements a model with the "random effects" feature. The lines declaring a `random_effects_vector` play a role similar to `init_vector` in earlier examples, except that the estimation method for random effects variables works differently (and much more slowly). The file `pheno.pin` includes initial values for the two random effects vectors declared in `pheno.tpl`.

How do I write R code to run ADMB?

The examples `simplePBS` and `vonb` both contain R files (`.r`) that illustrate the process. We focus here on the `vonb` example. Display 1 shows the Report Section in the model template. It writes variable names (preceded by $) and variable values. Running the executable file produces the report file `vonb.rep` listed in Display 2. This file has "PBS format", defined in the package PBSmodelling. Think of it as an R list object with named components.

Display 1. The Report Section in `vonb.tpl`.

```
REPORT_SECTION
  report << "$Linf"   << endl;
  report << Linf      << endl;
  report << "$K"      << endl;
  report << K         << endl;
  report << "$t0"     << endl;
  report << t0        << endl;
  report << "$sigma"  << endl;
  report << sigma     << endl;
  report << "$fval"   << endl;
  report << fval      << endl;
  report << "$age"    << endl;
  report << age       << endl;
  report << "$size"   << endl;
  report << size      << endl;
  report << "$spred"  << endl;
  report << spred     << endl;
```

Display 2. The report file `vonb.rep` produced by running `vonb.exe`. To fit in the space available on this page, the vectors `size` and `spred` are truncated. The file represents an R list with named components.

```
$Linf
57.2689
$K
0.164044
$t0
0.152865
$sigma
0.492146
$fval
-3.34367
$age
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
$size
 7.36 14.3 21.8 27.6 31.5 35.3 39 41.1 43.8 45.1 ...
$spred
 7.43029 14.9707 21.3702 26.8015 31.4111 35.3233 ...
```

Display 3. The R source file `vonb.r`. In the R console, the command `source("vonb.r")` initializes `PBSadmb` from a file `Adopts.txt` (presumably available and correct), makes `vonb.exe` from `vonb.tpl`, generates the plot in Figure 2, and compares results computed independently by ADMB and R.

```
# Initialize
require(PBSmodelling); require(PBSadmb); initAD("Adopts.txt")

# Make and run the file
makeAD("vonb"); runAD("vonb");

# Use PBSmodelling functions to read and unpack the report;
vonb <- readList("vonb.rep"); unpackList(vonb);

# Plot the data, all from ADMB
plot(age,size); lines(age,spred,col="red",lwd=2);

# Check the calculations in R

spredR <- Linf*(1-exp(-K*(age-t0)));
nobs   <- length(age);
fvalR  <- nobs*log(sigma) + sum((spredR-size)^2)/(2.0*sigma^2)

cat("Functions values (ADMB & R):\n");
cat(fval,"   ",fvalR,"\n")

cat("Predictions (ADMB & R):\n");
cat(spred,"\n");
cat(spredR,"\n");
```

Once you understand the relationship between the Report Section in Display 1 and the report file in Display 2, examine the R code in Display 3. It produces the plot in Figure 2, based entirely on data exported from the ADMB model. The PBSmodelling functions `readList` and `unpackList` produce R variables with the same names as corresponding variables in the template file.

The code in Display 3 illustrates the use of `PBSadmb` functions `initAD`, `makeAD`, and `runAD`. This guide ends with a complete list of functions currently available in the package.

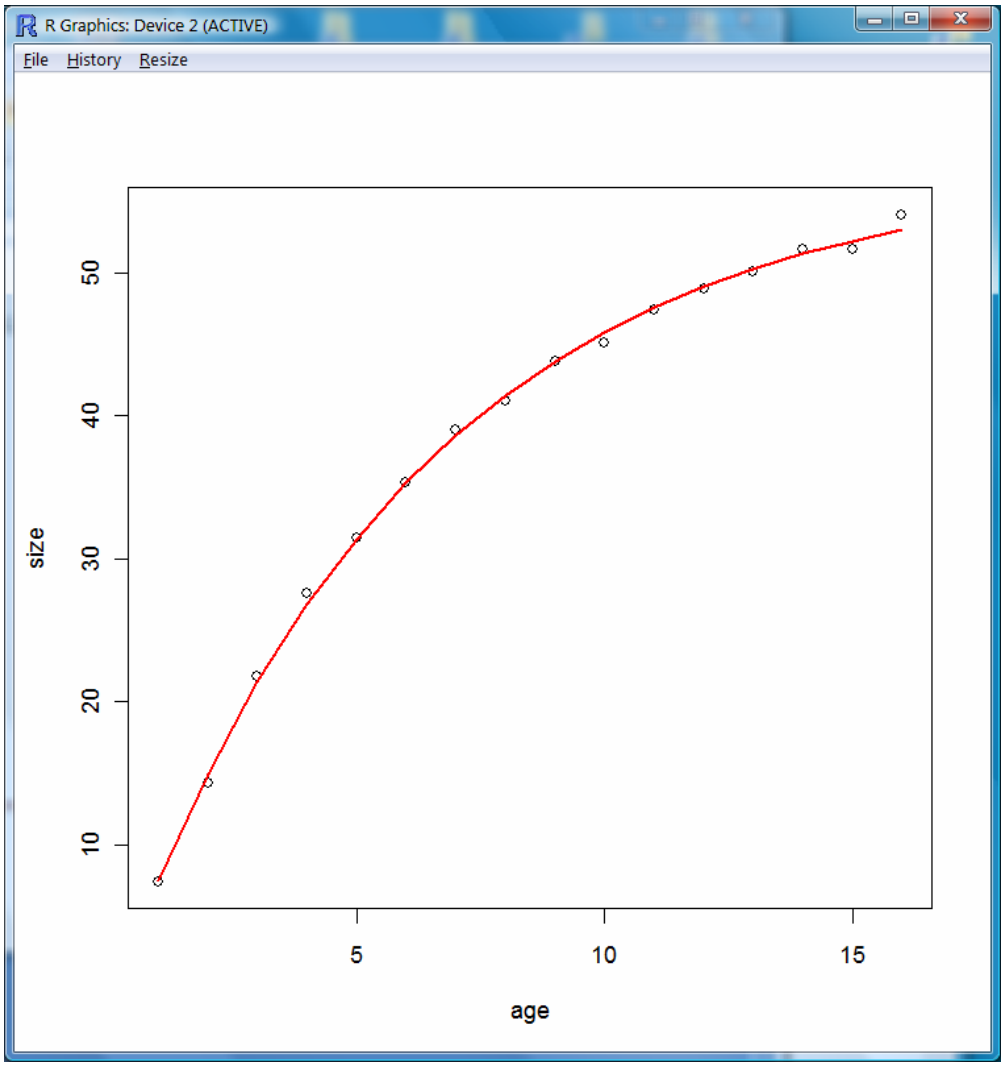

Figure 2. Plot of data points and a fitted von Bertalanffy growth curve, obtained by sourcing the R code in Display 3. The data portrayed here come entirely from the ADMB model. The source code compares these numbers with independent calculations in R.

This page intentionally left blank.