User's Guide to the R Package `PBSadmb`

by Jon Schnute and Rowan Haigh
Pacific Biological Station, Nanaimo, BC, Canada
November 2009

1. Introduction

Perhaps only a small minority of R users know about the powerful software package ADMB (AD Model Builder, http://admb-project.org/) released into the public domain in 2009. It provides a remarkably efficient tool for estimating parameters and their uncertainty, based on complex nonlinear statistical models. Its effectiveness stems partly from the use of *automatic differentiation* (AD, also called *algorithmic differentiation*) to compute the gradient of an objective function to be minimized. It includes robust algorithms for modal estimation and Markov chain Monte Carlo (MCMC) sampling from Bayesian posterior distributions. Other common inference methods, such as asymptotic covariances and 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. Thanks to a generous grant from the Gordon and Betty Moore Foundation (http://www.moore.org/), the ADMB Project (http://admb-project.org/) acquired rights to the software and began releasing it to the public domain in 2008. At the time of writing this report, the release has nearly been completed (http://admb-project.org/community/public-domain). Many people worked hard to make this possible, and we thank all of them for their efforts. An authoritative history of ADMB remains to be written, but it would make a very colourful story for an ambitious historian of computer science who has a lively sense of humour. It involves a cast of remarkable personalities who know how to develop serious scientific tools while having a great deal of fun. Not by accident, some of the spin-off packages bear the names of New Zealand wines, such as Coleraine (http://fish.washington.edu/research/coleraine/).

The R software environment easily accommodates external programs. R packages routinely include C/C++ code, and the packaging system automatically compiles the code for all supported operating systems. More generally, R can connect to a wide range of software written independently. For example, the open source program `ggobi` (http://www.ggobi.org/, "*Good pictures force the unexpected upon us*") allows users to visualize high dimensional data in a number of creative ways. This software runs independently from R, but the package `rggobi` allows R users to think of it as just another R application. Commands in R allow you to do anything that you could otherwise do with `ggobi`. To make things work, a user may need to install `ggobi` in the operating system of choice before installing the R package `rggobi`.

ADMB 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++. The author (Dave 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 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. Another command `runAD` runs the executable file with specified arguments.

The native interface to ADMB differs slightly among operating systems. For example, a Windows platform uses DOS batch files, whereas a Linux system uses bash scripts. Although this doesn't create any serious problems, it does require a bit of adjustment when moving from one system to another. The R platform, available on Windows, Linux, and MacOS X, offers a common interface that appears the same, regardless of the operating system. We have designed `PBSadmb` to take advantage of this fact. Consequently, a user who interacts with ADMB via R sees exactly the same interface on every platform.

`PBSadmb` allows a user to enter all ADMB commands in an R terminal, rather than a DOS or bash terminal. Furthermore, because R is now the language of choice, commands to ADMB can be integrated with R commands in the same script file. We introduce standards that make it possible to preserve variable names between R scripts and ADMB template files. A single R script can use ADMB to make an executable file, generate an MCMC sample, and draw a `pairs` plot of the results.

Although `PBSadmb` has the primary goal of accessing ADMB via R scripts, we also provide a Graphical User Interface (GUI) that greatly facilitates ADMB model development. New users may find it particularly helpful for editing code, testing it rapidly, and inspecting results (such as MCMC simulations). The GUI gives links to help files and examples that illustrate key aspects of ADMB model development. Use of the GUI is, however, entirely optional, and experienced users of ADMB and R may confine their applications of `PBSadmb` entirely to R script files. Even they might still find the GUI useful for configuring the software to run properly.

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 (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`) existed prior to `PBSadmb` on the Comprehensive R Archive Network (CRAN, http://cran.r-project.org/). We use Google Code web sites to maintain a source code archive for each of our packages. See http://code.google.com/p/pbs-software/ for links to all of them. In particular, http://code.google.com/p/pbs-admb/ has the source code and other information about `PBSadmb`.

Arguably, this package should have been called `pbsADMB` to put the proper emphasis on the role of ADMB. We have chosen, however, to preserve the naming style of our other

packages because they tend to be closely linked. For example, `PBSadmb` uses numerous functions from `PBSmodelling` and extends many of the programming goals of that earlier package. We encourage users to try all of our packages, or at least read their descriptions.

If you are an R user who wants the freedom to build arbitrarily complex statistical models, we believe you'll find this package an invaluable tool. Although ADMB was motivated by problems in fisheries science, professionals in many other fields (such as economics, finance, medicine, genetics, physics, and chemistry) will likely be surprised, if not astonished, at its power. We've written this package to help make ADMB transparent, useful, and available to a much wider audience than its traditional core in fishery science.

2. Using `PBSadmb`

To use `PBSadmb`, you first need to install it properly by the detailed procedure in Appendix A. As suggested in the introduction, this involves the two R packages `PBSadmb` and `PBSmodelling`, as well as ADMB itself. A C/C++ compiler is also required, which needs special installation on a Windows platform, but may come automatically as part of Linux or MacOS X. The GUI also requires a suitable choice of text editor.

Because this package applies to R, we assume that our readers have at least some familiarity with R itself and the standard methods of installing packages from the CRAN repository. If you are new to ADMB, you need to know that a typical project has a file prefix (*) and three associated files to hold the code (*.tpl), input data (*.dat), and initial parameter values (*.pin).

We illustrate the use of ADMB by considering a very simple estimation problem for the familiar von Bertalanffy growth curve:

(1) $y_i = L_\infty[1-e^{-K(a_i-t_0)}]+\sigma\varepsilon_i$, where $i=1,\ldots,n$.

This formula calculates observed lengths y_i from observed ages a_i and a vector (L_∞,K,t_0,σ) of four unknown parameters. The residuals ε_i in (1) are assumed to be independent normal random variables with mean 0 and standard deviation 1. From the density function for a normal distribution, the negative log likelihood ℓ for this model is:

(2) $$\ell(L_\infty,K,t_0,\sigma \mid a_1,\ldots,a_n,y_1,\ldots,y_n) = n\log\sigma + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-z_i)^2,$$

where the predicted length z_i at age a_i is

(3) $$z_i(a_i;L_\infty,K,t_0,\sigma) = L_\infty[1-e^{-K(a_i-t_0)}].$$

We drop an additive constant in (2) that does not affect the analysis. The notation emphasizes that we regard ℓ as a function of the parameters for fixed values of the data.

If the ADMB prefix for this project is `vonb`, then the three text files `vonb.tpl`, `vonb.dat`, `vonb.pin` would contain, respectively:

- the code for ℓ in (2),
- the data (a_i, y_i) for $i = 1,\ldots,n$, and
- initial values of the parameters $(L_\infty, K, t_0, \sigma)$.

This operational framework motivates the scripting language developed in `PBSadmb`, which includes the following commands (some mentioned previously):

`convAD`	convert *.tpl to *.cpp,
`compAD`	compile *.cpp to a binary object,
`linkAD`	link the binary object with ADMB libraries and create an executable file,
`makeAD`	convert, compile, and link to make an executable file,
`runAD`	run an ADMB executable with specified command line arguments,
`showArgs`	show all possible command line arguments for an ADMB executable,
`runMC`	run an ADMB executable in MCMC mode,
`plotMC`	plot the results of an MCMC simulation,
`editAD`	edit text files for the current project in the text editor,
`readRep`	read one of the standard reports generated by an ADMB executable,
`startLog`	start a log file (*.log) of ADMB activity,
`appendLog`	append to a log file of ADMB activity,
`cleanAD`	remove files created by ADMB that tend to proliferate in the working directory,
`convOS`	convert text files to the format for the operating system (windows or unix).

These commands illustrate the functions available in `PBSadmb`. For a complete list, see Appendix C. The database `ADMBcmd` contains an archive of scripts used to perform ADMB commands with various compilers on various operating systems, as described in Appendix B.

3. The `PBSadmb` GUI

As we have emphasized, `PBSadmb` principally defines a scripting language for interacting with ADMB. However, the package ADMB itself is quite complex, and new users might find it rather intimidating. Even experienced users like us sometimes forget key details needed to accomplish certain tasks. For this reason we offer a GUI that greatly facilitates ADMB model development. In our own workshops, we have found it an invaluable tool for educational purposes.

The GUI (Figure 1) 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. As mentioned earlier, we give special support to MCMC samples with plots that allow a user to inspect the sampled chain. The widgets “Thin and “Col” (for “Columns”) enable a user to thin the current chain and select variables for plotting.

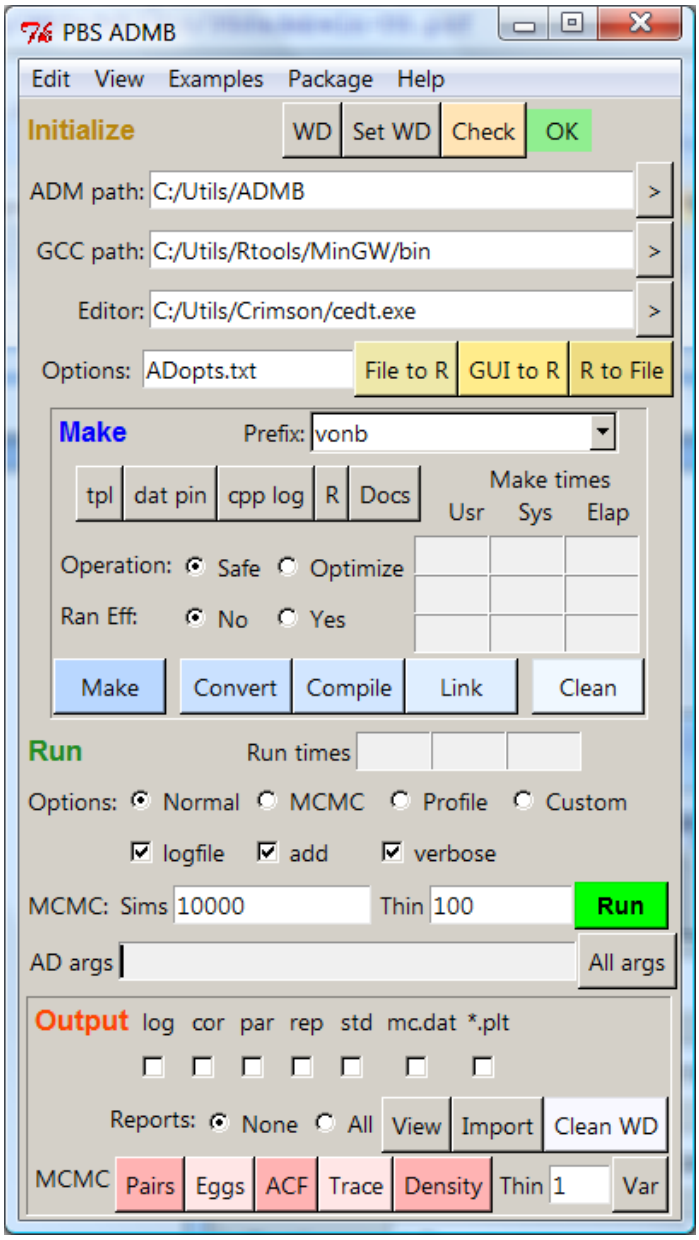

Figure 1. The graphical user interface (GUI) in `PBSadmb`, generated by the R command `admb()` on a Windows platform.

Buttons labelled ">" 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.

Experienced ADMB users know that ADMB leaves many "footprints" as 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 (by typing manually 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.

After you've successfully installed `PBSadmb`, 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.

4. ADMB in action

If you're like us, whenever you install a software package, you immediately want to see it do something. `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 GUI menu. If you click one of them (the file prefix), the program will load all related files into your 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.

When the GUI copies text files to the working directory, they automatically get converted (via the function `convOS`) to the correct format for the operating system, with line endings <CR><LF> in Windows and <LF> in Unix. Sometimes ADMB fails when the input files have a format inappropriate for the OS. (We encountered this problem with the ADMB command `tpl2rem`, called by our function `convAD`.) If you move files across platforms, remember that

conversion might be necessary. Linux users probably have the native OS commands `todos` and `fromdos` for this purpose. Windows users can get similar utilities from the Internet.

We encourage new users to explore the examples 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`.

`simple_mc`, a variant of `simple`, can give a Bayesian posterior sample of the parameters. The GUI allows you to perform a run with "MCMC" options (the number of simulations and the thinning frequency). You can then view results visually with plots generated from the "Output" section of the GUI.

`simple_pbs`, a variant of `simple_mc`, 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 `simple_pbs.r` performs four tasks:

- making the executable file `simple_pbs.exe` (in Windows) or `simple_pbs` (in Linux) from `simple_pbs.tpl`,
- running the executable file,
- loading the data from `simple_pbs.rep` into R, while preserving variable names, and
- producing a standard regression plot for the data exported imported from `simple_pbs.rep`.

`vonb`, similar to `simple_pbs`, implements the estimation problem posed by equations (1)–(3) in Section 2. 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`.

5. R scripts to run ADMB

The examples `simplePBS` and `vonb` both contain R files (`*.r`) that illustrate the use of R scripts to code ADMB analyses in R. 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.

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 functions `readList` and `unpackList` from `PBSmodelling` produce R variables with the same names as corresponding variables in the template file, given the structure of the Report Section in Display 1. This technique represents a standard in `PBSadmb` for writing ADMB template code to ensure variables with identical names and values in the R environment. *Just write the Report Section to export both names and values*, as illustrated in Display 1.

Display 1. The Report Section in `vonb.tpl`. It generates a report file `vonb.rep` that contains both variable names and values for easy import into the R environment. This technique ensures variables with common names and values in both ADMB and R.

```
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 << "$y"       << endl;
  report << y          << endl;
  report << "$ypred"   << endl;
  report << ypred      << endl;
  report << "$mcnames" << endl;
  report << "Linf K t0 sigma LK fval" << endl;
  report << "$mcest"   << endl;
  report << Linf << " " << K << " " << t0 << " " << sigma << " "
    << LK << " " << fval << endl;
```

Display 2. The report file `vonb.rep` produced by running `vonb.exe`. To fit in the space available on this page, the vectors `y` and `ypred` have been 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
$y
 7.36 14.3 21.8 27.6 31.5 35.3 39 41.1 43.8 45.1 ...
$ypred
 7.43029 14.9707 21.3702 26.8015 31.4111 35.3233 ...
$mcnames
Linf K t0 sigma LK fval
$mcest
57.2689 0.164044 0.152865 0.492146 9.39464 -3.34367
```

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); readADopts("Adopts.txt")

# Make and run "vonb.exe"
makeAD("vonb"); runAD("vonb");

# Read and unpack the report;
# i.e., create R variables with the same names used in "vonb.tpl"
vonb <- readList("vonb.rep"); unpackList(vonb);

# Plot the data
plot(age,y); lines(age,ypred,col="red",lwd=2);

# Check the calculations in R
ypredR <- Linf*(1-exp(-K*(age-t0)));
nobs   <- length(age);
fvalR  <- nobs*log(sigma) + sum((ypredR-y)^2)/(2.0*sigma^2)

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

cat("Predictions (ADMB & R):\n");
cat(ypred,"\n");
cat(ypredR,"\n");
```

The code in Display 3 can be supplemented by two simple commands:

```
> runMC("vonb",100000,100)
> plotMC("vonb")
```

to give the Bayesian posterior scatter plot shown in Figure 3. The first line runs 100,000 simulations, thinning to keep only 1 of every 100 results. The second line plots the 1,000 points that result from this calculation. The following lines from the Report Section (Display 1) play a key role in producing Figure 3:

```
report << "$mcnames" << endl;
report << "Linf K t0 sigma LK fval" << endl;
report << "$mcest" << endl;
report << Linf << " " << K << " " << t0 << " " << sigma << " "
  << LK << " " << fval << endl;
```

by communicating the names of the variables preserved in the MCMC simulation (`$mcnames`) and their values at the posterior mode (`$mcest`). These provide the graph with variable names along the diagonal and coordinates of the mode, shown as a red point corresponding to the minimum ℓ (`fval`).

The results in Figure 3 can also be obtained directly from the GUI. Furthermore, `plotMC` can also draw a variety of plots, as indicated by the five coloured buttons along the bottom line of the GUI. Try them!

Acknowlegements

Our intrepid programmer, Alex Couture-Beil, greatly assisted the development of `PBSadmb` by improving its integration with `PBSmodelling`. In some cases, this meant changing `PBSmodelling` to enhance support for external packages. We thank all members of the ADMB community for their efforts on the open source project, particularly Dave Fournier, John Sibert, Mark Maunder, and Johnoel Ancheta. Ian Taylor helped us experiment with early conversions of `PBSadmb` from Windows to Linux. Saang-Yuan Hyun conducted tests and made a number of helpful suggestions. Andy Edwards generously contributed funding to support programming expenses.

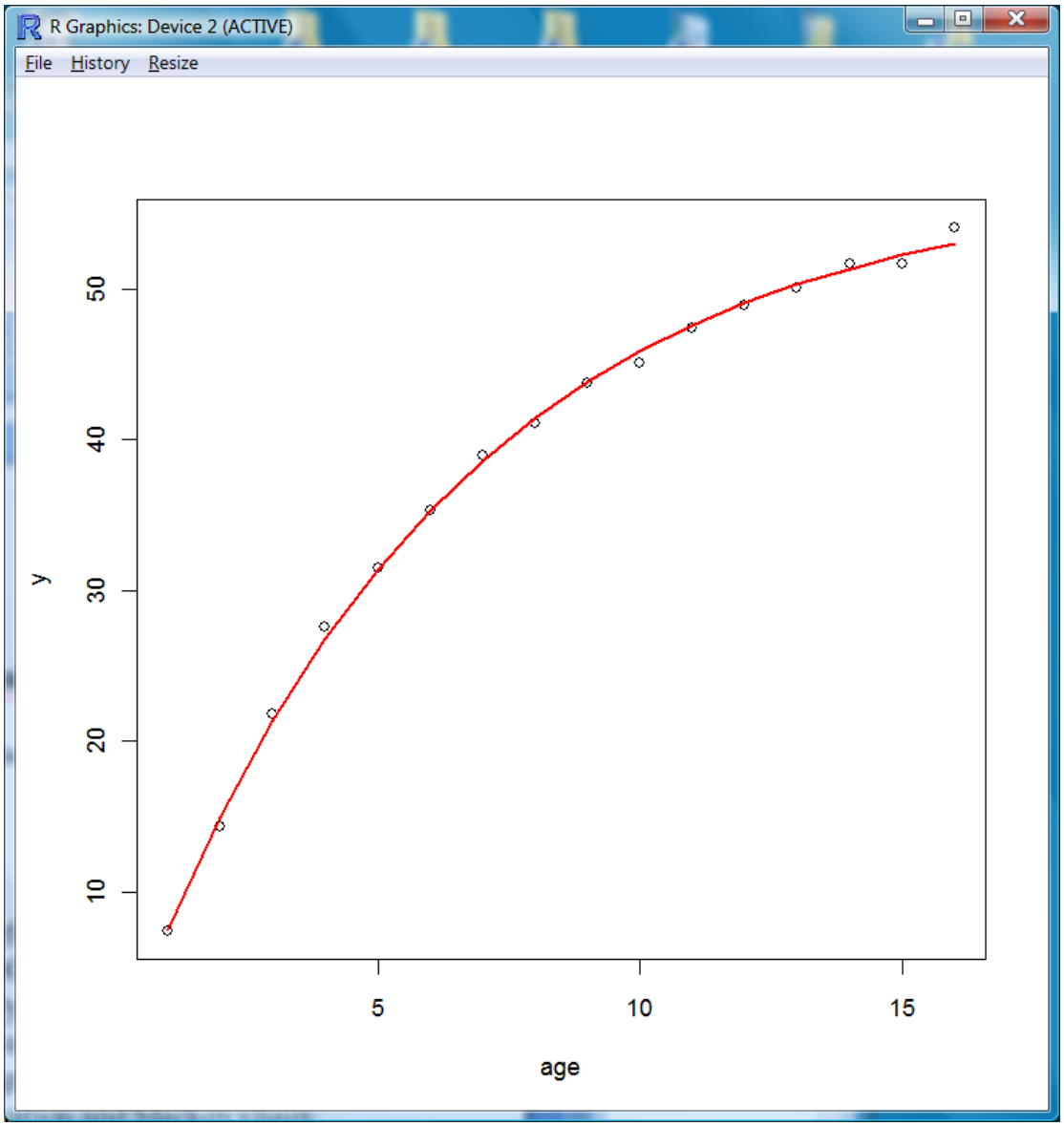

Figure 2. Plot of data points and a fitted von Bertalanffy growth curve, obtained by sourcing the R code in Display 3. The data portrayed here come entirely from the ADMB model. The source code compares these numbers with independent calculations in R.

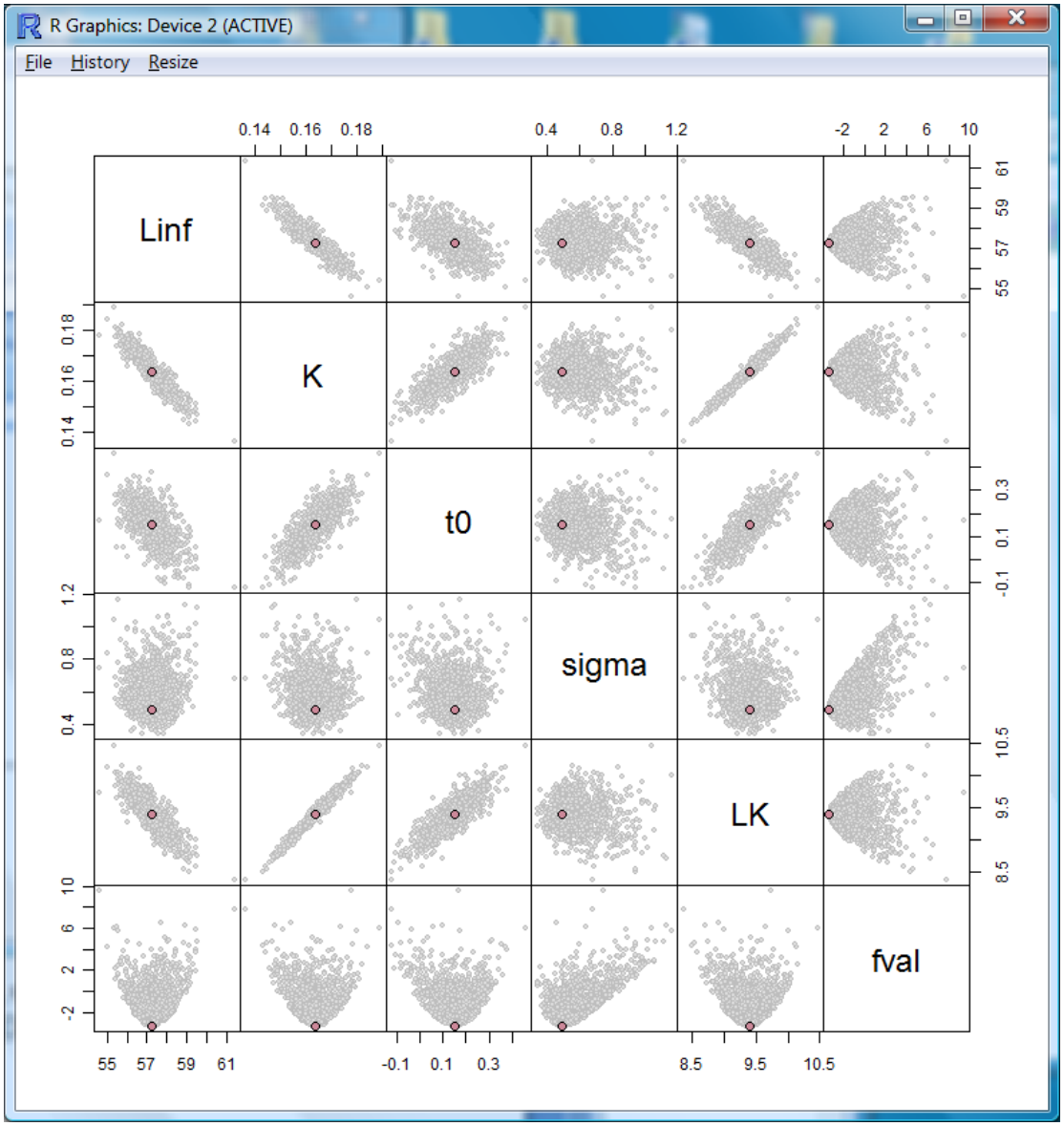

Figure 3. Scatter plot of 1,000 points from the `vonb` posterior distribution, generated by the code in `vonb.r` and the two additional commands: `runMC("vonb",100000,100); plotMC("vonb")`. The red point indicates the posterior mode, where `fval` is minimized. The variables labelled `Linf, K, t0, sigma, LK, and fval` represent L_∞, K, t_0, σ, the product $L_\infty K$, and ℓ, respectively.

Appendix A. Installing PBSadmb

As for most R packages, installation of PBSadmb is fairly easy. Unfortunately, this one requires other software as well, so please be patient while going through all the required steps. Essentially, you need to install R, PBSmodelling, 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 should have a working version of the interface in Figure 1. Here are the details.

Step 1. Install the current version of R for your operating system from a package manager or the CRAN web site http://cran.r-project.org/. We assume that you have enough familiarity with R to do this without difficulty. If you have a version of R already installed, update it to the current version (R 2.10.0 at the time of writing this report) if necessary.

Step 2. Run R and install current versions of the packages PBSmodelling and PBSadmb. Ideally, both of these should be available on CRAN, but we also plan to keep Windows binary (*.zip) and package source (*.tar.gz) files on our web sites:

http://code.google.com/p/pbs-modelling/ for PBSmodelling,
http://code.google.com/p/pbs-admb/ for PBSadmb.

In Windows, you can install packages from the R GUI, but on all systems you can use the command `install.packages()`.

Step 3 (Windows). If you have a Windows OS, go to the web site http://www.murdoch-sutherland.com/Rtools/, and download the file Rtools29.exe. Run this executable file, and install the R tools in a directory of your choice. In this example, we assume that 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)
```

This means that you're using g++ 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 PBSadmb.

Please note that we do not currently support the compilers by Borland or Microsoft. Any user keenly interested in such options can help by contributing to the database discussed in Appendix B.

Step 3 (Linux or MacOS X). If you have a Unix system, hopefully you already have compiler support for C/C++. To check this, open a bash window and type

```
g++ --version
```

Hopefully, you'll see a result like

```
g++ (Ubuntu 4.3.3-5ubuntu4) 4.3.3
```

You also need to know where the executable `g++` is located. On our Ubuntu system, it's in the directory `/usr/bin/`, but you may to go to the root directory (`cd /`) and run a command (`whereis g++`) to find the path.

Step 4. Obtain a good text editor that you can use for code development. On Windows, the 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, like the Crimson Editor (http://www.crimsoneditor.com/) or Tinn-R (http://www.sciviews.org/Tinn-R/). Ideally, use an editor that supports syntax highlighting and displays multiple files in a single window, with tabs to select among them.

On Linux systems, `gedit` seems to work reasonably well.

Step 5. Download the package ADMB for your OS, from the official web site http://admb-project.org/downloads. Currently, we support the following distributions:

Operating System	**File Description**	**File**
Windows	Windows MinGW GCC 3.4	admb-9.0.363-win32-mingw-gcc3.4.zip
Linux 32	Linux (32-bit) GCC 4.2	admb-9.0.363-intel-linux32-gcc4.2.zip
Linux 64	Linux (64-bit) GCC 4.2	admb-9.0.363-intel-linux64-gcc4.2.zip
Intel MacOS	Intel MacOS (10.4) for GCC 4.0	admb-9.0.202rc3-macos10.4-gcc4.0.zip

Once you have the relevant zip file, look through its contents to locate the directory that contains a few text files (including `LICENSE` and `README`) and subdirectories (including `bin`, `include`, and `lib`). Put all these files and directories in a directory of your choice, such as `C:\Utils\ADMB` for Windows or `/home/Waldo/ADMB` in Linux if your username happens to be `Waldo`. (Thus the home directory `~` is equivalent to `/home/Waldo`.)

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 three paths, always using the Unix syntax in which the forward slash `/` (rather than the Windows backslash `\`) separates subdirectories.

- The "ADM path" should be the path chosen in Step 5, such as `C:/Utils/ADMB` or `/home/Waldo/ADMB`. This directory should have the ADMB subdirectories `bin`, `include`, and `lib`.
- The "GCC Path" should be the path to `g++` in Step 3, such as `C:/Utils/Rtools/MinGW/bin` or `/usr/bin`.
- The "Editor" should be the complete path to executable file for the editor chosen in Step 4, such as `C:/Utils/Crimson/cedt.exe` or `/usr/bin/gedit`.

You can use the buttons labelled ">" to navigate to the appropriate directories or files.

Step 7. In the "Initialize" section, click the "Check" button. If everything is OK, you should see the green message "OK" in the adjacent text box. The red message "Fix" means that something essential can't be found on the paths you've specified. Either you haven't installed something correctly, or one of the paths is wrong.

When everything is OK, click "GUI to R" to save your specified options in the (hidden) R variable `.PBSadmb` that contains your specified options. It has class `PBSoptions`, defined in `PBSmodelling`. As usual, you can inspect it in the R console by typing its name `.PBSadmb`. Finally, click "R to File". This creates a file `Adopts.txt` in your current working directory that you can inspect with the text editor.

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.

Appendix B. ADMB scripts in `ADMBcmd`

As we have discussed, ADMB involves the three basic operations `convAD`, `compAD`, and `linkAD` to convert, compile, and link a template file. The software also offers two binary options: safe/optimized and normal/random effects. In theory, these operations and their options give $3 \times 2 \times 2 = 12$ combinations. However, conversion doesn't depend on the safe/optimized option and compilation doesn't depend on the choice normal/random effects. This leaves only the 8 possible commands to ADMB that are indexed in the table below:

Index	Step	Safe	Random Effects
1	convert		F
2	convert		T
3	compile	F	
4	compile	T	
5	link	F	F
6	link	T	F
7	link	F	T
8	link	T	T

Each of these commands can vary among operating systems and compilers. To achieve multilingual status, `PBSadmb` needs a database of the relevant 8 commands for each combination of operating system and compiler. This is contained in the data frame `ADMBcmd`, with column names `OS` (operating system), `Comp` (compiler), `Index` (index from the table above), `Step` (convert, compile, or link), `Safe` (safe mode when true, optimized when false), `RanEff` (random effects when true, normal when false), `Command` (coded version of the shell command to the operating system), and `Comment` (any additional information). You can view `ADMBcmd` with the R command:

```
> data(ADMBcmd); ADMBcmd
```

For reasons associated with R, various operating systems, and regular expressions, we use the character @ to denote environment variables in our coded commands, where

- `@prefix` is the current prefix,
- `@adHome` is the path to ADMB, chosen during installation Step 5 (Appendix A) and shown as "ADM path" in the GUI,
- `@ccPath` is the path to `g++` chosen during installation Step 3 (Appendix A) and shown as "GCC path" in the GUI.

For similar reasons, we use a backtick ` to denote quotation marks (rather than an apostrophe ' or double quotes ").

As an example of our encoding scheme, command #1 in Windows and Linux is, respectively,

```
`@adHome/bin/tpl2cpp.exe` @prefix
`@adHome/bin/tpl2cpp ` @prefix
```

where backticks are used in case the variable `@adHome` has embedded spaces. (`PBSadmb` does not allow a prefix to have embedded spaces.) Currently, our script management system distinguishes only between the options available for the R variable `.Platform$OS.type` which are `windows` or `unix`. Happily, these two options appear adequate to the support the ADMB versions listed in installation Step 5.

Users interested in support for other compilers can help us by expanding the database `ADMBcmd` to include more alternatives. The distribution of `PBSadmb` includes an Excel spreadsheet with our current scripts in the directory `admb_scripts`. For highly technical readers, we point out that command #2 requires special treatment, due to the implementation of the ADMB command `tpl2rem`.

Appendix C. Detailed `PBSadmb` documentation

C.1. Functions in ADMB

The list of functions in this section comes from the `PBSmodelling` command

```
> viewCode("PBSadmb",output=2)
```

Function	Description
admb	Start the PBS ADMB GUI
appendLog	Append Data to Log File
checkADopts	Check ADMB Options for Link Integrity
cleanAD	Clean ADMB-Generated Files from the Working Directory
compAD	Compile C Code
convAD	Convert TPL Code to CPP Code
convOS	Convert Text Files to Default OS Format
copyFiles	Copy System Files
editAD	Edit ADMB Files

editADfile	Edit a File
installADMB	Install windows admb binary (for gcc)
linkAD	Link Object Files to Make an Executable
makeAD	Make an Executable Binary File from a C File
makeADopts	Creates the ADMB Options List
parseCmd	Parse an Indexed ADMB Command
plotMC	Plot Results of MCMC Simulation
readADopts	Reads an ADMB Options List into Memory From a File
readRep	Read an ADMB Report into R Memory
runAD	Run an Executable Binary File
runMC	Run an Executable Binary File in MCMC Mode
showADargs	Show All Arguments for an ADMB Executable
startLog	Start a Log File
writeADopts	Writes the ADMB Options List from Memory to a File

C2. `PBSadmb` manual

The following pages show the standard R manual for `PBSadmb`, including help pages for all objects, a table of contents, and an index. This manual also appears on the CRAN web site: http://cran.r-project.org/web/packages/PBSadmb/index.html. (Or from CRAN's root, locate "Packages" and find "PBSmodelling".)

For a description of the method used to generate the pages that follow, see Appendix D.2 of the *PBSmodelling User's Guide* included with `PBSmodelling`.

This page intentionally left blank.

Package 'PBSadmb'

November 20, 2009

Version 0.51.42

Date 2009-11-20

Title PBS ADMB

Author Jon T. Schnute, Rowan Haigh

Maintainer Jon T. Schnute <Jon.Schnute@dfo-mpo.gc.ca>

Depends R (>= 2.7.0), PBSmodelling (>= 2.06)

Description R Support for ADMB (Automatic Differentiation Model Builder)

License GPL (>=2)

R topics documented:

admb . . . 20
ADMBcmd . . . 20
appendLog . . . 21
checkADopts . . . 22
cleanAD . . . 23
compAD . . . 23
convAD . . . 24
convOS . . . 25
copyFiles . . . 26
editAD . . . 27
editADfile . . . 27
installADMB . . . 28
linkAD . . . 28
makeAD . . . 29
makeADopts . . . 30
parseCmd . . . 31
plotMC . . . 31
readADopts . . . 32
readRep . . . 33
runAD . . . 34
runMC . . . 35
showADargs . . . 36
startLog . . . 36
writeADopts . . . 37

admb *Start the PBS ADMB GUI*

Description

Start up the PBS GUI for running ADMB.

Usage

```
admb(prefix="", wdf="admbWin.txt", optfile="ADopts.txt")
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").
wdf	string name of the *window description file* that creates the GUI.
optfile	string name of options file (usually in user's working directory).

Author(s)

Rowan Haigh, Pacific Biological Station, Nanaimo BC, Canada

See Also

makeADopts

ADMBcmd *Database of ADMB Command Scripts*

Description

Command scripts for ADMB's convert, compile, and link routines

Usage

```
data(ADMBcmd)
```

Format

A data frame with the following 8 variables:

OS operating system

Comp C++ compiler type

Index index that indicates convert, compile or link with options: safe or optimize, random effects or normal.

Step description of processing step (convert, compile, or link)

Safe logical: if TRUE use safe mode; if not, use optimise mode.

RanEff logical: if TRUE use random effects model; if not, use normal model.

Command the command suitable for specified combination of Step, Safe, and RanEff

Comment comment about the command, if any

Details

This database represents a compilation of ADMB scripts for various operating systems and compilers. A user's project normally starts with a template file, named with a prefix to denote the project and a standard suffix `.tpl`. This file must go through three processing steps: conversion to C/C++ code, compilation by a specified compiler, and linking with ADMB libraries.

The reulting command depends on the operating system, compiler, processing step, and two binary options (safe/optimized; normal/random effects). In principle, the three processing steps and two binary options give 3x2x2=12 possibilities. However, conversion doesn't depend on the "safe/optimized" choice, and compilation doesn't depend on "normal/random effects". This reduction leaves only 8 possibilities, specified by an `index` in the range 1:8.

A variable in a `Command` string is designated by the prefix character `@`. We use this for convenient string substitution by `parseCmd`, the function that translates database strings into actual ADMB commands.

The subdirectoy `.../ADMB/scripts` in the installed package contains an Excel spreadsheet, used as the source file for this database. Currently, our database is incomplete, and we heartily encourage the ADMB community to make contributions for additional operating systems and compilers.

Source

Jon T. Schnute, Pacific Biological Station, Nanaimo BC

See Also

`parseCmd`

---

appendLog *Append Data to Log File*

Description

Append summary information or output to a previously created log file.

Usage

```
appendLog(prefix, lines)
```

Arguments

`prefix`	string name prefix of the ADMB project (e.g., `"vonb"`).
`lines`	data to append to `'prefix'.log`).

Value

No explicit value reurned. Appends data into a log file `'prefix'.log`.

Note

A wrapper function that can be called from a GUI exists as `.win.appendLog`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

startLog, editADfile

---

checkADopts *Check ADMB Options for Link Integrity*

Description

Check that .ADopts has all required components and that links point to actual files on the hard drive.

Usage

```
checkADopts(opts=getOptions(.PBSadmb), check=c("admpath","gccpath","editor"),
            warn=TRUE, popup=FALSE)
```

Arguments

opts	ADMB options values.
check	components of .ADopts to check.
warn	logical: if TRUE, print the results of the check to the R console.
popup	logical: if TRUE, display program location problems in a popup GUI.

Value

Boolean value where TRUE indicates all programs were located in the specified directories and FALSE if at least one program cannot be found. The returned Boolean scalar has two attributes:
warn - named list of test results, and
message - named vector of test results.

Note

A wrapper function that can be called from a GUI exists as .win.checkADopts.

Author(s)

Rowan Haigh, Pacific Biological Station, Nanaimo BC, Canada

See Also

makeADopts,readADopts

cleanAD *Clean ADMB-Generated Files from the Working Directory*

Description

Detects files in the working directory with the specified `prefix` and removes them all save those with the suffix `.tpl`, `.dat`, and `.pin`.

Usage

```
cleanAD(prefix)
```

Arguments

`prefix`	string name prefix of the ADMB project (e.g., `"vonb"`).

Details

Aside from potential garbage files with the specified `prefix`, other files associated with ADMB are detected. Also files `*.tmp` and `*.bak` are displayed. Calling `cleanAD` invokes the hidden function `.cleanUp`, which creates a GUI menu of the potential garbage files. The user can select whichever files s/he wishes for disposal.

Value

Returns nothing. Invokes a GUI menu of potential garbage files.

Note

A wrapper function that can be called from a GUI exists as `.win.cleanAD`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

`makeAD`, `runAD`, `readRep`

compAD *Compile C Code*

Description

Compile C++ code in `'prefix'.cpp` to create a binary object file `'prefix'.o`.

Usage

```
compAD(prefix, raneff=FALSE, safe=TRUE, logfile=TRUE, add=TRUE,
       verbose=TRUE, comp="GCC")
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").
raneff	logical: use the random effects model, otherwise use the normal model (currently does not influence the compile stage, but the argument is preserved here for future development).
safe	logical: if TRUE, use safe mode with bounds checking on all array objects, otherwise use optimized mode for fastest execution.
logfile	logical: if TRUE, create a log file of the messages from the shell call.
add	logical: if TRUE, append shell call messages to an exsiting log file.
verbose	logical: if TRUE, report the shell call an its messages to the R console.
comp	string: compiler to use - "GCC" is only currently supported

Details

This function uses the C++ comiler declared in .ADopts. If logfile=TRUE, any errors will appear in 'prefix'.log. If verbose=TRUE, they will appear in the R console.

Value

Invisibly returns the shell call and its messages.

Note

A wrapper function that can be called from a GUI exists as .win.compAD.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

convAD, linkAD, makeAD

---

convAD *Convert TPL Code to CPP Code*

Description

Convert code in 'prefix'.tpl to C++ code in 'prefix'.cpp.

Usage

```
convAD(prefix, raneff=FALSE, logfile=TRUE, add=FALSE,
       verbose=TRUE, comp="GCC")
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").
raneff	logical: if TRUE, use the random effects model executable tpl2rem.exe, otherwise use the normal model executable tpl2cpp.exe.
logfile	logical: if TRUE, create a log file of the messages from the shell call.
add	logical: if TRUE, append shell call messages to an exsiting log file.
verbose	logical: if TRUE, report the shell call an its messages to the R console.
comp	string: compiler to use - "GCC" is only currently supported

Details

This function invokes the ADMB command `tpl2cpp.exe` or `tpl2rem.exe`, if `raneff` is `FALSE` or `TRUE` respectively. If `logfile=TRUE`, any errors will appear in `'prefix'.log`. If `verbose=TRUE`, they will appear in R console.

Value

Invisibly returns the shell call and its messages.

Note

A wrapper function that can be called from a GUI exists as `.win.convAD`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

`compAD`, `linkAD`, `makeAD`

---

convOS *Convert Text Files to Default OS Format*

Description

Convert text files to the default format of the operating system.

Usage

```
convOS(inam, onam = inam, path = getwd() )
```

Arguments

`inam`	string vector of names specifying files to be converted to the format of the operating system.
`onam`	string vector of name specifying the output files (the default overwrites the input file).
`path`	string specifying the path where the input files are located (defaults to current working directory).

Value

Text file(s) formatted in accordance with standards of the operating system.

Note

This function essentially executes a `readLines` command followed by a call to `writeLines`.

Author(s)

Rowan Haigh, Pacific Biological Station, Nanaimo BC

See Also

`copyFiles`, `.addQuotes`

copyFiles *Copy System Files*

Description

Copy files with specified prefixes and suffixes from one location to another.

Usage

```
copyFiles(prefix, suffix=NULL, dir0=getwd(), dir1=getwd(), ask=TRUE)
```

Arguments

prefix	string scalar/vector of potential file prefixes.
suffix	string scalar/vector of potential file suffixes.
dir0	source directory from which to copy files.
dir1	destination directory to copy files to.
ask	logical: if TRUE, popup boxes will prompt the user for every instance that a file will be overwritten.

Details

This function uses R's `list.files` and `file.copy` functions. The pattern recognition tends not to work when given the wildcard character `*`; however, the user may use this character, and the code will interpret it.

Value

Invisibly returns a Boolean vector with names of files that have been copied or not.

Author(s)

Rowan Haigh, Pacific Biological Station, Nanaimo, BC

See Also

editAD

editAD *Edit ADMB Files*

Description

Edit files associated with specified prefix and suffixes.

Usage

```
editAD(prefix, suffix=c(".tpl",".cpp",".log"))
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").
suffix	string scalar/vector specifying one or more suffixes.

Value

Invisibly returns Boolean vector with elements TRUE if files exist, FALSE if they do not.

Note

A wrapper function that can be called from a GUI exists as .win.editAD.

This function explicitly uses the editor chosen for PBSadmb. PBSmodelling has another function openFile that uses Windows file associations or an application specified with setPBSext.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

editADfile, makeADopts

---

editADfile *Edit a File*

Description

Edit a file using the text editor specified in .ADopts.

Usage

```
editADfile(fname)
```

Arguments

fname	string name of file in current working directory (or elsewhere if path delimited by / or \).

Value

Returns Boolean: TRUE if file exists, FALSE if it does not.

Note

This function explicitly uses the editor chosen for PBSadmb. PBSmodelling has another function openFile that uses Windows file associations or an application specified with setPBSext.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

editAD, makeADopts

installADMB *Install windows admb binary (for gcc)*

Description

Only applicable for Windows: Downloads and installs the windows ADMB binary for gcc. ADMB is installed under PBSadmb's library directory under R.

Usage

```
installADMB()
```

Value

The path where ADMB was installed.

linkAD *Link Object Files to Make an Executable*

Description

Links the binary object file 'prefix'.o to the ADMB libraries and produces the executable file 'prefix'.exe.

Usage

```
linkAD(prefix, raneff=FALSE, safe=TRUE, logfile=TRUE, add=TRUE,
       verbose=TRUE, comp="GCC")
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").
raneff	logical: use the random effects model, otherwise use the normal model.
safe	logical: if TRUE, use safe mode with bounds checking on all array objects, otherwise use optimized mode for fastest execution.
logfile	logical: if TRUE, create a log file of the messages from the shell call.
add	logical: if TRUE, append shell call messages to an exsiting log file.
verbose	logical: if TRUE, report the shell call an its messages to the R console.
comp	string: compiler to use - "GCC" is only currently supported

Details

This function uses the C++ comiler declared in .ADopts. If logfile=TRUE, any errors will appear in 'prefix'.log. If verbose=TRUE, they will appear in the R console.

Value

Invisibly returns the shell call and its messages.

Note

A wrapper function that can be called from a GUI exists as `.win.linkAD`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

`convAD`, `compAD`, `makeAD`

---

makeAD *Make an Executable Binary File from a C File*

Description

Essentially a wrapper function that calls in sequence: `convAD`, `compAD`, and `linkAD`.

Usage

```
makeAD(prefix, raneff=FALSE, safe=TRUE, logfile=TRUE, verbose=TRUE)
```

Arguments

`prefix`	string name prefix of the ADMB project (e.g., `"vonb"`).
`raneff`	logical: use the random effects model, otherwise use the normal model.
`safe`	logical: if `TRUE`, use safe mode with bounds checking on all array objects, otherwise use optimized mode for fastest execution.
`logfile`	logical: if `TRUE`, create a log file of the messages from the shell call.
`verbose`	logical: if `TRUE`, report the shell call an its messages to the R console.

Details

This function uses the C++ comiler declared in `.ADopts`. If `logfile=TRUE`, any errors will appear in `'prefix'.log`. If `verbose=TRUE`, they will appear in the R console.

Value

Returns nothing. The three functions called by `makeAD` each return the shell call and its messages.

Note

A wrapper function that can be called from a GUI exists as `.win.makeAD`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

`convAD`, `compAD`, `linkAD`, `cleanAD`

makeADopts *Creates the ADMB Options List*

Description

Creates a global list object detailing the pathways to the ADMB directory, the GCC bin, and the user's preferred text editor.

Usage

```
makeADopts(admpath, gccpath, editor)
```

Arguments

admpath	explicit path to the user's ADMB directory.
gccpath	explicit path to the user's GCC bin (C-compiler) directory.
editor	explicit path and program to use for editing text.

Value

Creates a global, hidden list object called `.ADopts`.

Note

A wrapper function that can be called from a GUI exists as `.win.makeADopts`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

`makeADopts`, `writeADopts`

parseCmd *Parse an Indexed ADMB Command*

Description

Parse an indexed ADMB command line for a specified `index`, operating system (`os`), and compiler (`comp`). The result depends on the project `prefix`, the path (`admpath`) to the ADMB home directory, and the path (`gccpath`) to the C++ compiler. Within the database, variables are denoted by leading `@` characters.

Usage

```
parseCmd(prefix, index, os=.Platform$OS, comp="GCC", admpath="", gccpath="")
```

Arguments

prefix	prefix for the ADMB project.
index	index that indicates one of eight possibilities related to three processing steps (convert, compile, link) and options: safe or optimize, random effects or normal.
os	operating system
comp	C++ compiler description
admpath	explicit path for the ADMB home directory.
gccpath	explicit path for the C++ bin directory.

Value

Character string, the ADMB command from ADMBcmd corresponding to the specified index, prefix, and system paths.

Author(s)

Rowan Haigh, Pacific Biological Station, Nanaimo BC

See Also

ADMBcmd

---

plotMC *Plot Results of MCMC Simulation*

Description

Plot results of an ADMB MCMC simulation using various plot methods.

Usage

```
plotMC(prefix, act="pairs", pthin=1, useCols=NULL)
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").
act	string scalar: action describing plot type (current choices: "pairs", "eggs", "acf", "trace", and "dens").
pthin	numeric scalar indicating interval at which to collect records from the .mc.dat file for plotting.
useCols	logical vector indicating which columns of .mc.dat to plot.

Note

A wrapper function that can be called from a GUI exists as .win.plotMC. Use the PBSadmb GUI to explore these plots easily.

Author(s)

Rowan Haigh, Pacific Biological Station, Nanaimo BC, Canada

See Also

runMC, showADargs

---

readADopts *Reads an ADMB Options List into Memory From a File*

Description

Reads ADMB options into a global, hidden list object called `.ADopts` from an ASCII text file using `PBSmodelling::readList`).

Usage

```
readADopts(optfile="ADopts.txt")
```

Arguments

`optfile`	string name of an ASCII text file containing ADMB options information.

Value

No values returned. Reads the ADMB options into the list object `.ADopts`.

Note

A wrapper function that can be called from a GUI exists as `.win.readADopts`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

makeADopts,writeADopts

---

readRep *Read an ADMB Report into R Memory*

Description

Import ADMB-generated report files into R's memory using the names of the report files to name the R-objects.

Usage

```
readRep(prefix, suffix=c(".cor",".rep",".std",".mc.dat"), global=FALSE)
```

Arguments

`prefix`	string name prefix of the ADMB project (e.g., `"vonb"`).
`suffix`	string scalar/vector specifying one or more suffixes.
`global`	logical: if `TRUE`, save the imported reports as objects to global environment using the same names as the report files.

Details

If the report object is one of `c(".cor", ".std", ".mc.dat")`, the report object is a data frame, otherwise it is a string vector. Multiple report objects are returned as a list of objects. A single report object is returned as the object itself.

This function attempts to detect the file format from a number of possibilities. For example, if the file has the special format recognized by PBSmodelling, then the function returns a list with named components. The example `vonb` included with this package shows how to write the template to get consistent variable names between ADMB and R. See the User's Guide for complete details.

Value

Invisibly returns the list of report objects. If only one report is imported, a single report object is returned.

Note

A wrapper function that can be called from a GUI exists as `.win.readRep`.

Author(s)

Rowan Haigh, Pacific Biological Station, Nanaimo BC, Canada

See Also

`editADfile`, `.win.viewRep`

---

runAD *Run an Executable Binary File*

Description

Run the executable binary file `'prefix'.exe` that was created by `makeAD`.

Usage

```
runAD(prefix, argvec="", logfile=TRUE, add=TRUE, verbose=TRUE)
```

Arguments

`prefix`	string name prefix of the ADMB project (e.g., `"vonb"`).
`argvec`	string scalar/vector of arguments appropriate for the executable `'prefix'.exe`.
`logfile`	logical: if `TRUE`, create a log file of the messages from the shell call.
`add`	logical: if `TRUE`, append shell call messages to an exsiting log file.
`verbose`	logical: if `TRUE`, report the shell call an its messages to the R console.

Details

This function typically reads the two files `'prefix'.dat` and `'prefix'.pin`, although in same cases one or both of these files may not be necessary.

If `logfile=TRUE`, output (including error messages, if any) will appear in `'prefix'.log`. If `verbose=TRUE`, it will appear in the R console.

Value

Invisibly returns the results of the shell call.

Note

A wrapper function that can be called from a GUI exists as `.win.runAD`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

`runMC`, `makeAD`, `cleanAD`

---

runMC *Run an Executable Binary File in MCMC Mode*

Description

Run the executable binary file `'prefix'.exe`, created by `makeAD`, to generate MCMC simulations.

Usage

```
runMC(prefix, nsims=2000, nthin=20, outsuff=".mc.dat",
      logfile=FALSE, add=TRUE, verbose=TRUE)
```

Arguments

`prefix`	string name prefix of the ADMB project (e.g., `"vonb"`).
`nsims`	numeric scalar indicating number of MCMC simulations to perform.
`nthin`	numeric scalar indicating the sampling rate or thinning of the `nsims` MCMC simulations to report.
`outsuff`	string name suffix of the MCMC output data file.
`logfile`	logical: if `TRUE`, create a log file of the messages from the shell call.
`add`	logical: if `TRUE`, append shell call messages to an exsiting log file.
`verbose`	logical: if `TRUE`, report the shell call an its messages to the R console.

Details

This function runs `'prefix'.exe` twice, first with the arguments `-mcmc 'nsims' -mcsave 'nthin'` and second with the argument `-mceval`. By default, output goes to the file `'prefix'.mc.dat`, although a user can specify a different output suffix.

To see this function in action, use the PBSadmb GUI with the example `vonb` or `simpleMC`.

Value

Invisibly returns the results of the shell call.

Note

A wrapper function that can be called from a GUI exists as `.win.runMC`.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

runAD, makeAD, cleanAD

---

showADargs *Show All Arguments for an ADMB Executable*

Description

Show all arguments available for an ADMB executable in the default text editor.

Usage

```
showADargs(prefix, ed=TRUE)
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").
ed	logical: if TRUE, write the ADMB arguments to a file and view them with the text editor, else display the arguments on the R console.

Value

Invisibly returns the argument list.

Note

A wrapper function that can be called from a GUI exists as .win.showADargs.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

editADfile, runAD

startLog *Start a Log File*

Description

Start a log file by removing any previous version and appending header information.

Usage

```
startLog(prefix)
```

Arguments

prefix	string name prefix of the ADMB project (e.g., "vonb").

Value

No explicit value reurned. Writes header lines into a log file 'prefix'.log.

Note

A wrapper function that can be called from a GUI exists as .win.startLog.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

appendLog, editADfile

writeADopts *Writes the ADMB Options List from Memory to a File*

Description

Writes the global ADMB options list to a file in 'PBS' format (see PBSmodelling::writeList).

Usage

```
writeADopts(optfile="ADopts.txt")
```

Arguments

optfile	string name of the intended output file.

Value

Returns opts invisibly. Writes the options list object to an ASCII file.

Note

A wrapper function that can be called from a GUI exists as .win.writeADopts.

Author(s)

Jon T. Schnute, Pacific Biological Station, Nanaimo BC, Canada

See Also

`makeADopts`, `readADopts`

Index

∗Topic **IO**
admb, 20
copyFiles, 26
∗Topic **character**
convOS, 25
∗Topic **datasets**
ADMBcmd, 20
∗Topic **data**
checkADopts, 22
makeADopts, 30
readADopts, 32
writeADopts, 37
∗Topic **file**
appendLog, 21
convOS, 25
editAD, 27
editADfile, 27
installADMB, 28
readRep, 33
showADargs, 36
startLog, 36
∗Topic **hplot**
plotMC, 31
∗Topic **interface**
compAD, 23
convAD, 24
linkAD, 28
makeAD, 29
runAD, 34
runMC, 35
∗Topic **list**
checkADopts, 22
makeADopts, 30
readADopts, 32
writeADopts, 37
∗Topic **manip**
cleanAD, 23
parseCmd, 31
readRep, 33
∗Topic **programming**
compAD, 23
convAD, 24
linkAD, 28
makeAD, 29
runAD, 34
runMC, 35
∗Topic **utilities**
copyFiles, 26

admb, 20
ADMBcmd, 20, *31*
appendLog, 21, *37*

checkADopts, 22
cleanAD, 23, *30*, *34, 35*
compAD, 23, *25*, *29, 30*
convAD, 24, *24*, *29, 30*
convOS, 25
copyFiles, 26, *26*

editAD, *26*, 27, *28*
editADfile, *22*, 27, *27*, *33*, *36, 37*

installADMB, 28

linkAD, *24, 25*, 28, *30*

makeAD, *23–25*, 29, *29*, *34, 35*
makeADopts, *20*, *22*, *27, 28*, 30, *30*, *32*, *37*

parseCmd, *21*, 31
plotMC, 31

readADopts, *22*, 32, *37*
readRep, *23*, 33
runAD, *23*, 34, *35, 36*
runMC, *32*, *34*, 35

showADargs, *32*, 36
startLog, *22*, 36

writeADopts, *30*, *32*, 37