PBSadmb 0.69: User's Guide

Jon T. Schnute, Rowan Haigh, and Alex Couture-Beil

Fisheries and Oceans Canada
Science Branch, Pacific Region
Pacific Biological Station
3190 Hammond Bay Road
Nanaimo, British Columbia
V9T 6N7

2015

User's Guide Formatted as a Canadian Technical Report of Fisheries and Aquatic Sciences

Fisheries and Oceans Canada | Pêches et Océans Canada

Canada

© Her Majesty the Queen in Right of Canada, 2015

Revised from Cat. No. Fs97-6/2674E ISSN 0706-6457

Last update: Feb 5, 2015

Correct citation for this publication:

Schnute, J.T., Haigh, R., and Couture-Beil, A. 2015. PBSadmb 0.69: user's guide formatted as a Canadian Technical Report of Fisheries and Aquatic Sciences: ii + 36 p. Last updated Feb 5, 2015

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Schnute, J.T., Haigh, R., and Couture-Beil, A. 2015. PBSadmb 0.69: user's guide formatted as a Canadian Technical Report of Fisheries and Aquatic Sciences: ii + 36 p. Last updated Feb 5, 2015.

This report describes the R package **PBSadmb**, an R wrapper for the powerful software package ADMB (AD Model Builder, http://admb-project.org/) released into the public domain in 2009. The ADMB framework constitutes 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. It also supports other common inference methods, such as asymptotic covariances and likelihood profiles. With the package **PBSadmb**, 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. For example, a single R script can use ADMB to make an executable file, generate an MCMC sample, and draw a `pairs` plot of the results. More generally, **PBSadmb** allows a user to treat ADMB as just another R application, with the same interface in all supported operating systems (currently Windows, Linux, or Mac OS).

User's Guide to the R Package PBSadmb

by Jon T. Schnute, Rowan Haigh, and Alex Couture-Beil
Pacific Biological Station, Nanaimo, BC, Canada

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. Several releases have now been completed, and many people have worked hard to make this possible. We thank all of them for their efforts and mention a few names in the Acknowledgements below. 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/) and Awatea (http://code.google.com/p/pbs-awatea/). Currently, **PBSadmb** supports ADMB version 10 or higher.

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 produce 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. They might also 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/science/facilities-installations/pbs-sbp/index-eng.html). 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 the main PBS Software page 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. If you use the **PBSadmb** GUI (Steps 5 and 6, Appendix A), you also need a suitable choice of text editor.

To facilitate the setup of paths, the user can supply a path file (with a default name "`ADpaths.txt`"). This two-column text file contains basic information to the three paths used by **PBSadmb**, where the first column specifies the R variable (`admbpath`, `gccpath`, `editor`) and the second column specifies the full path used by the R variable (the columns being delimited by space). Unix users need not specify the second path (`gccpath`) to a C++ compiler. A typical path file might look like:

```
admbpath  "C:/Apps/admb"
gccpath   "C:/Apps/mingw"
editor    "C:/Apps/Notepad++/notepad++.exe"
```

A simple call: `readADpaths("ADpaths.txt")` copies these paths to the options manager, which is used by other **PBSadmb** functions. You should also verify that these paths are valid using `checkADopts()`.

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`).

If the ADMB prefix for this project is `vonb`, then the three input text files would contain:

- `vonb.tpl` the code for ℓ in (2),
- `vonb.dat` the data (a_i, y_i) for $i = 1,\ldots,n$, and
- `vonb.pin` 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:

`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;
`showADargs`	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 package also includes a set of ADMB example projects (see Section 5).

3. R scripts to run ADMB

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.

The example projects `simple_pbs` and `vonb` (Section 5) 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, which you can copy to your working directory by issuing this command from the R Console:

```
copyFiles("vonb", srcdir=system.file("examples",package="PBSadmb"),
    dstdir=getwd(), ask=TRUE)
```

Assuming that you have project files in your working directory and paths available in the options manager (see Section 2), you can edit `vonb` files from the R Console by issuing the command:

```
editAD(prefix="vonb",suffix=c(".tpl",".r"))
```

Table 1 shows the Report Section in the model template `vonb.tpl`. It writes variable names (preceded by `$`) and variable values. Running the executable file produces the report file `vonb.rep` listed in Table 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 Table 1 and the report file in Table 2, examine the R code in Table 3. It produces the plot in Figure 1, 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 Table 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 Table 1.

Table 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;
```

Table 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
```

Table 3. The R source file `vonb.r`. In the R console, the command `source("vonb.r")` initializes **PBSadmb** using the function `setupAD` (using a file `ADpaths.txt`, presumably available and correct), makes `vonb.exe` from `vonb.tpl`, generates the plot in Figure 1, and compares results computed independently by ADMB and R.

```
# Initialize
require(PBSmodelling); require(PBSadmb); setupAD()

# 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,pch=20,cex=2,col="blue",xlab="Age",ylab="Length (mm)");
lines(age,ypred,col="red",lwd=3);

# 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");
```

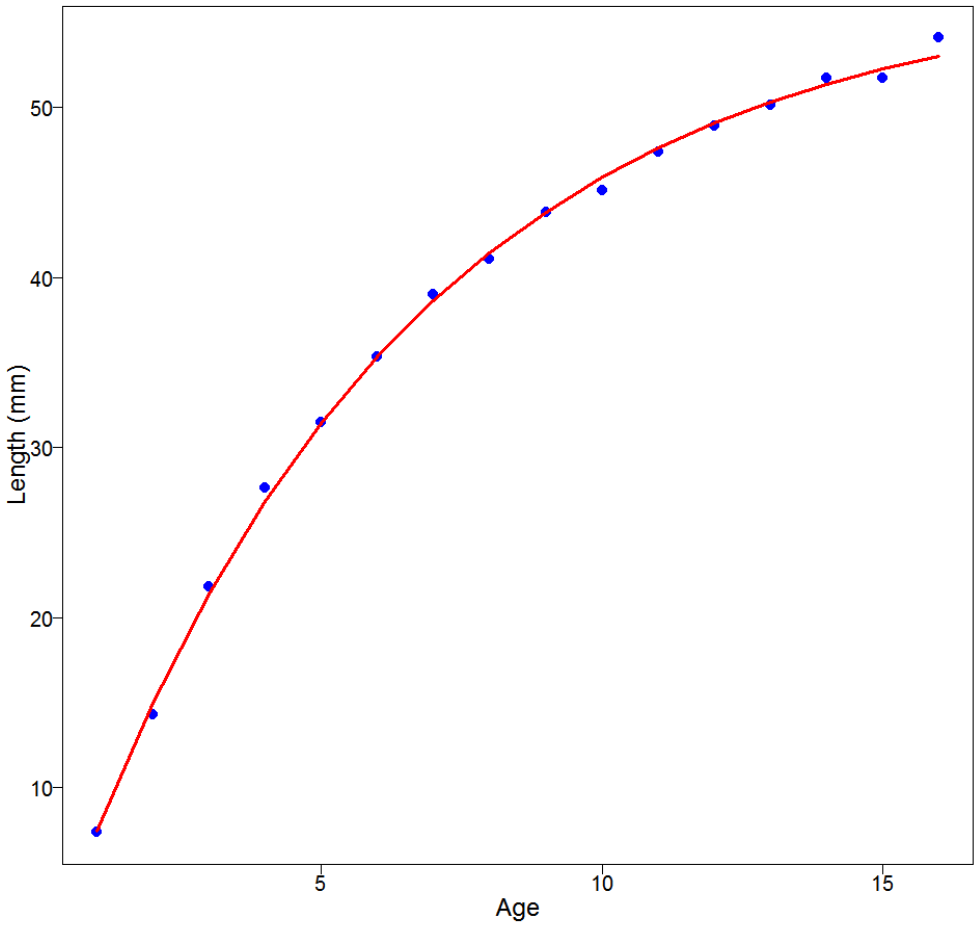

Figure 1. Plot of data points and a fitted von Bertalanffy growth curve, obtained by sourcing the R code in Table 3. The data portrayed here come entirely from the ADMB model. The source code compares these numbers with independent calculations in R.

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

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

to give the Bayesian posterior scatter plot shown in Figure 2. 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 (Table 1) play a key role in producing Figure 2:

```
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`).

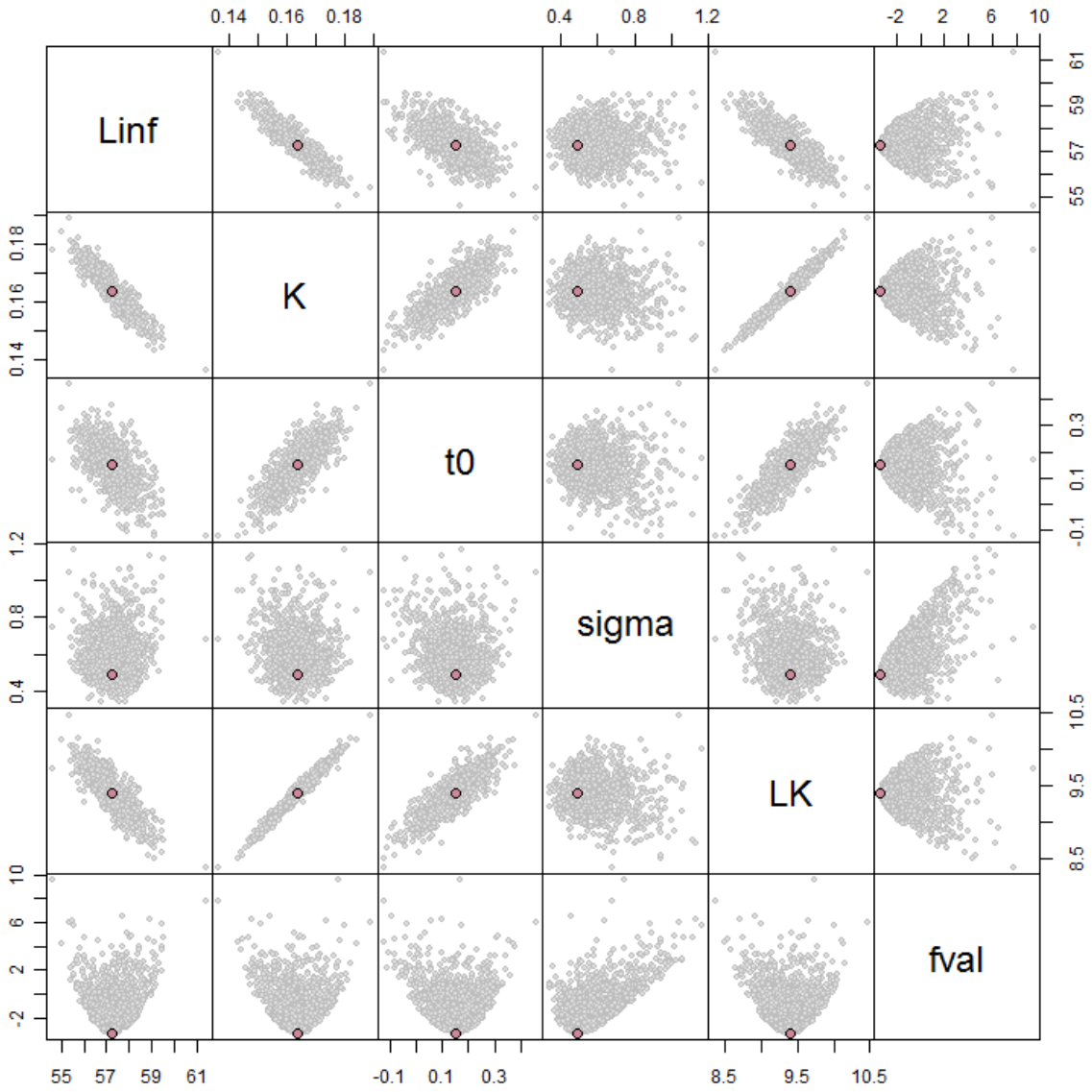

Figure 2. Scatter plot of 1,000 points from the `vonb` posterior distribution, generated by the code in `vonb.r` and two additional commands: `runMC("vonb",100000,100); plotMC("vonb")`. The red point indicates the posterior mode, where `fval` is minimized. The variables labelled `Linf, K, t0, sigma, LK, and fval` represent L_∞, K, t_0, σ, the product $L_\infty K$, and ℓ, respectively.

The results in Figure 2 can also be obtained directly from a handy GUI (Section 4). 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!

4. 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 sometimes 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.

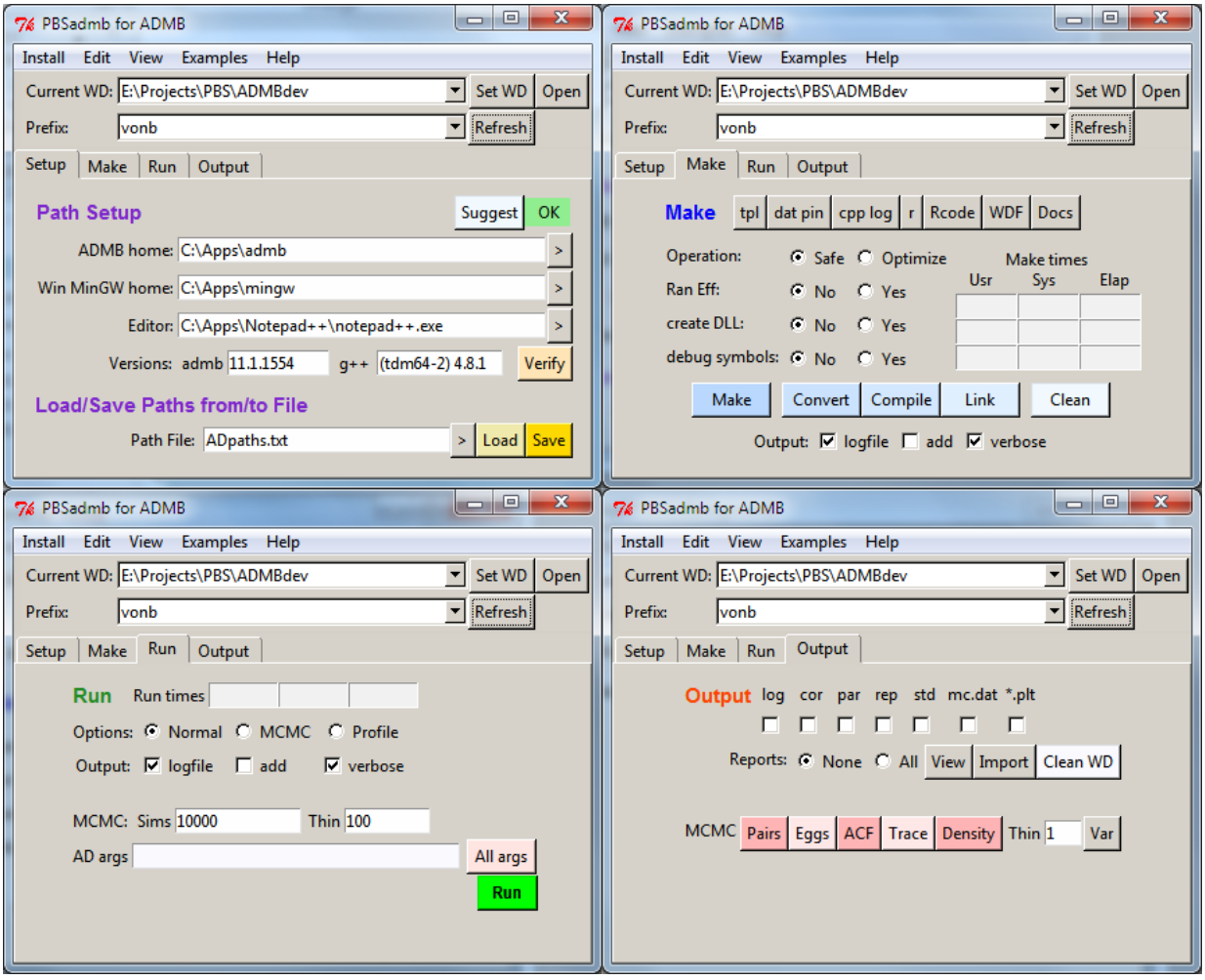

Figure 3. The graphical user interface (GUI) in **PBSadmb**, generated by the R command `admb()` on a Windows platform. The window makes use of `notebook` tabs: from left to right – **Setup**, **Make**, **Run**, and **Output**.

The GUI (Figure 3) allows a user to explore all aspects of ADMB model development. The interface emphasizes four distinct phases:

- **Setup** the package with appropriate paths , check that they make sense, and save them in a file with the default name `ADpaths.txt`. Additionally, a backup file called `ADopts.txt` saves paths and version numbers detected for ADMB and g++.
- **Make** the executable file for a chosen prefix, with options between "Safe" and "Optimize" 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. Turn off "Verbose" while producing MCMC samples.
- 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 "Var" (for "Variables") enable a user to thin the current chain and select variables for plotting.

Buttons labelled ">" in the **Setup** section allow a user to browse for available paths to installations of ADMB, a C++ compiler, and an editor . To facilitate the setup of paths, the user can supply a path file (with a default name "`ADpaths.txt`"). This two-column text file contains basic information to the three paths, where the first column specifies the R variable used by the GUI (`admbpath`, `gccpath`, `editor`) and the second column specifies the full path used by the R variable (the columns being delimited by space). Unix users need not specify the second path (`gccpath`) to a C++ compiler.

The **Make** section provides a few options when building an executable project file. Grey text boxes in this section show the computational 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. <Install> provides

links to the ADMB website that detail how to install ADMB on your operating system. <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 3. <Help> points to manuals in the package, online resources, and this User's Guide.

5. Example projects

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. On the R Console, you can see these example files using:

```
list.files(system.file("examples",package="PBSadmb"))
```

You can also copy project files to your working directory by issuing the following command:

```
copyFiles("vonb", srcdir=system.file("examples",package="PBSadmb"),
    dstdir=getwd(), ask=TRUE)
```

For those using the GUI, click <Examples> on the 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 focus particularly on `vonb`.

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:

- makes the executable file `simple_pbs.exe` (in Windows) or `simple_pbs` (in Linux) from `simple_pbs.tpl`,
- runs the executable file,
- loads the data from `simple_pbs.rep` into R, while preserving variable names, and
- produces 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 3. 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 this 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`.

Acknowledgements

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. Andrew Edwards contributed funding to support programming expenses.

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**, a C/C++ compiler (which normally comes installed on a Unix platform), ADMB, and a text editor suitable for writing templates and viewing reports. Then you need to run R, load **PBSadmb**, and give it some configuration (path) information. At this point, you can either run command line functions or launch the GUI (graphical user interface, Figure 3).

STEP 1. Install R

Install the current version of R for your operating system from a package manager or the CRAN web site http://cran.r-project.org/. For fast downloads, choose a mirror near you. 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-3.2.0 at the time of revising this report) if necessary.

STEP 2. Install PBSmodelling and PBSadmb

Run R and install current versions of the packages **PBSmodelling** and **PBSadmb**. Ideally, both of these should be available on CRAN, but we also maintain the current source code on our web sites:

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

Additionally, the most recent versions of the Windows binaries (`*.zip`) can be found at:

https://drive.google.com/folderview?id=0B2Bkic2Qu5LGeWtUNi1CQ04yVEk&usp=drive_web

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

STEP 3 (Windows). Install C++ compiler and ADMB

Next you need to install a suitable C++ compiler for Windows (MinGW), along with a corresponding version of the ADMB package. In the current implementation of **PBSadmb**, we provide a link to explicit installation instructions on the ADMB web site:

http://www.admb-project.org/tools/admb-tools-for-windows

If you are using the GUI, go to the Menu and press <Install><ADMB Tools for Windows>.

After installation of ADMB and MinGW, correct paths must also be specified in the configuration file (by default `ADpaths.txt`).

STEP 3 (Linux or MacOS X). Install C++ compiler and ADMB

If you have a Unix system, hopefully you already have compiler support for C/C++. To check this, open a bash terminal window and type: `g++ --version`

You should see something like

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

You may also need to know where the executable `g++` is located, though we assume it occurs somewhere on your path. Using the `Sys.genv()["PATH"]` command in R, the path is likely:

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

On an Ubuntu system, you can also go to the root directory (`cd /`) and run a command (`whereis g++`) to find the path.

In addition to the compiler, you also need to obtain ADMB for your system. Installation instructions are provided on the ADMB website:

Linux: http://www.admb-project.org/documentation/install/admb-installation-linux
or in GUI: <Install><ADMB Installation Linux>

Mac OS: http://www.admb-project.org/documentation/install/admb-installation-macos-package-installer
or in GUI: <Install><ADMB Installation Mac XCode>

The most reliable method will probably come from downloading the full source distribution and following the instructions to build and install the package from source. That way, you guarantee that the binary version of ADMB matches the compiler on your system. You also need to find the path to the installed version of ADMB and specify this in your `ADpaths.txt` configuration file.

STEP 4. Choose a text editor

Obtain a good text editor that you can use for code development. On Windows, the Notepad will work, but much better options are available. Currently, we recommend Notepad++, available free at http://notepad-plus-plus.org/. Other colleagues have used 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. In Mac OS X, you might consider using `TextWrangler` available free at http://www.barebones.com/products/textwrangler/download.html.

STEP 5. Launch the GUI

If running the GUI, start R in an empty working directory. Then type these two commands into the R console:

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

The **PBSadmb** GUI should appear at the **Setup** tab, probably with a red field **Fix** and a warning message about missing files. You can use the GUI to set the three required paths (two in Unix),

preferably using Unix syntax in which the forward slash / (rather than the Windows backslash \) separates subdirectories.

- The "ADMB home" should refer to the directory with the installed ADMB binary version. Typically it includes a few files (such as `LICENSE` and `README.txt`) and several subdirectories, including `bin`, `include`, and `lib`.
- The "Win MinGW home" should be the path discussed in Step 3. It should include a `bin` subdirectory with the file `g++.exe`. On Unix systems, this GUI entry is ignored because the C++ compiler and system tools like `sed` already occur on the `PATH`.
- The "Editor" should be the complete path to executable file for the editor chosen in Step 4, such as `C:/Utils/Npp/notepad++.exe`.

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

Step 6. Verify your paths

Click the "Verify" button. If all paths are valid, the red **Fix** should be replaced by a green **OK**. If **Fix** persists, then 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 "Save" to save your options in a text file in your current working directory, with the default name `ADpaths.txt`. You can inspect it with the text editor. Potentially, you can have multiple path files that point to different text editors or different compilers (e.g., 32 bit or 64 bit).

In the future, when you issue the R command `admb()` with this working directory, the file `ADpaths.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 path files for projects in different directories.

An additional file called `ADopts.txt` acts as a back-up file and is only generated when the user presses "Save" to save `ADpaths.txt`. The back-up file contains information on the three paths and version numbers for ADMB and `g++`. If the back-up file `ADopts.txt` is available in a user's working directory and `ADpaths.txt` is missing for some reason, the GUI uses information from the back-up file when a new R session is opened. It is preferable to use `ADpaths.txt` over `ADopts.txt` for tracking software paths.

Appendix B. ADMB scripts

As we have discussed, ADMB involves the three basic operations `convAD`, `compAD`, and `linkAD` to convert, compile, and link a template file, respectively. These are implemented in ADMB with scripts written as `batch` files for Windows or `bash` files for Unix. The scripts generally have the file prefix as an argument, along with other arguments that determine compilation options like safe or optimized code. The **Make** tab of the **PBSadmb** GUI has radio buttons that support four binary alternatives – safe or optimized code, normal or random effects model, create `.exe` or `.dll`, and include debugging symbols or not.

Appendix C. PBSadmb documentation

C.1. Functions in PBSadmb

The list of functions in this section comes from the **PBSmodelling** command `viewCode`; however, this only works for R>=3.2.0.

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

Function	Description
admb	Start the PBSadmb GUI for ADMB
alisp	List objects in temporary work environment (PBSadmbEnv)
appendLog	Append data to log file
atcall	Call objects from temporary work environment
atget, atput	Get/Put objects from or to temporary work environment
atprint	Print objects in temporary working environment
checkADopts	Check ADMB options for path 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
linkAD	Link object files to make an executable
makeAD	Make an executable binary file from a C file
plotMC	Plot results of MCMC simulation
readADopts	Read an ADMB options list into memory from a file
readADpaths	Read ADMB paths from a text file
readRep	Read an ADMB report into R memory
runAD	Run an executable binary file
runMC	Run an executable binary file in MCMC mode
saveADpaths	Read ADMB paths from a text file
setADMBpath	Create the ADMB options list
setADMBver	Detect the version numbers for ADMB and `g++`
setupAD	Set up paths for PBSadmb
showADargs	Show all arguments for an ADMB executable
startLog	Start a log file
suggestPath	Suggest a path to a specified program
writeADopts	Write the ADMB options list from memory to a file

C.2. PBSadmb reference manual

The automatically generated reference manual for **PBSadmb** appears on the CRAN web site: http://cran.r-project.org/web/packages/PBSadmb/index.html. For a description of the method used to generate similar reference pages for a document appendix, see Appendix D.2 of the *PBSmodelling User's Guide* included with **PBSmodelling**.