AD Studio

Statistical modelling in ADMB and TMB
Version 1.0 (2018-01-19)

Arni Magnusson

This is the manual for AD Studio version 1.0.
The latest edition of the manual is available at:
http://admb-project.org/tools/adstudio

Copyright © 2018 Arni Magnusson

AD Studio is an aggregate of the following software components:

- ADMB 12.0, released under the BSD License.
  Source code: http://ftp.admb-project.org/
- Emacs 25.3.1, released under the GPL.
  Source code: ftp://ftp.gnu.org/gnu/emacs/
- Emacs ADMB Mode 12.0-0, released under the Simplified BSD License.
  Source code: https://github.com/admb-project/admb/blob/master/contrib/emacs
- Emacs AUCTEX 12.1, released under the GPL.
  Source code: ftp://ftp.gnu.org/gnu/auctex/
- Emacs Markdown Mode 2.3, released under the GPL.
  Source code: https://github.com/jrblevin/markdown-mode
- Emacs Speaks Statistics (ESS) 17.11, released under the GPL.
  Source code: http://ess.r-project.org/downloads/ess/
- Emacs TMB Mode 3.4, released under the GPL.
  Source code: https://github.com/kaskr/adcomp/blob/master/emacs
- Rtools 3.4 (GCC and GDB), released under the GPL.
  Source code: ftp://ftp.gnu.org/gnu/gcc/ and ftp://ftp.gnu.org/gnu/gdb/

Table of Contents

1 Preamble 1
- 1.1 Credit 1
- 1.2 Simplified Emacs 1
- 1.3 History and name 1
- 1.4 System requirements 1

2 AD Studio overview 2
- 2.1 Statistical modelling 2
- 2.2 Software components 2
- 2.3 Installation 3

3 ADMB tutorial 5
- 3.1 Create a working copy of `simple` 5
- 3.2 Build, run, and view the results 6
- 3.3 Debug 9

4 TMB tutorial 14
- 4.1 Create a working copy of `mini` 14
- 4.2 Build, run, and view the results 14
- 4.3 Debug 15

5 User interface 16
- 5.1 Menu 16
- 5.2 Toolbar 18
- 5.3 Keybindings 19

6 Hints and tips 22
- 6.1 Help pages 22
- 6.2 Files 22
- 6.3 Cut, copy, paste 22
- 6.4 Undo/redo 22
- 6.5 Comment/uncomment 23
- 6.6 Secondary window 23
- 6.7 Alt key 23
- 6.8 Vi keybindings 23

7 Configuration 24
- 7.1 Personal `.emacs` file 24
- 7.2 Custom startup 24

8 Troubleshooting 25
- 8.1 General usage 25
- 8.2 Configuration 25

9 References 27

1 Preamble

1.1 Credit

AD Model Builder (ADMB) was written by David Fournier at Otter Research in Canada, and Template Model Builder (TMB) was written by Kasper Kristensen at DTU Aqua in Denmark.

1.2 Simplified Emacs

The main purpose of AD Studio is to make the convenient Emacs features of `admb-mode` and `tmb-mode` available to non-Emacs users. In other words, to disable the standard Emacs behavior.

Experienced Emacs users may prefer to ignore the AD Studio `.emacs` file, and simply install and load ADMB Mode (`admb.el`) and TMB Mode (`tmb.el`) like other Emacs packages. They are written as standard "major modes" that follow all Emacs mode conventions.

1.3 History and name

AD Studio is the unification of two previous software products, ADMB-IDE (Magnusson 2009) and TMB-IDE (Magnusson 2015). The 'AD' stands for automatic differentiation and 'Studio' is a commonly used synonym for IDE (integrated development environment).

1.4 System requirements

64-bit

AD Studio requires a 64-bit operating system. Windows 32-bit users can continue to use previous versions of ADMB-IDE (11.2) and TMB-IDE (1.6).

Rtools, R, TMB

Rtools, R, and TMB may already exist on the machine before AD Studio is set up. If Rtools is not already installed, the AD Studio installer can (optionally) do that. Users are expected to install R and the TMB package on their own, as these are frequently updated.

2 AD Studio overview

2.1 Statistical modelling

Developing statistical models in ADMB and TMB is an iterative process that consists of writing, compiling, testing, and debugging. AD Studio has been designed specifically to allow the user to perform these tasks more efficiently than in other working environments. For example, TMB model development involves working on C++ and R code side by side on the screen, but in some editors, such as RStudio, it is not possible to view two source code files at the same time.

GNU Emacs is a complex and powerful editor that comes with particularly good support for C++, R, LaTeX, backup/version control, and other useful features for statistical computing. The dedicated ADMB and TMB modes provide syntax highlighting, compilation, file manipulation, outline code navigation, templates, and smaller tools for developing models. Emacs users can download `admb-mode.el` and `tmb-mode.el` and start using them right away, after reading the commentary at the top of these files.

The problem with Emacs is that it requires considerable time to learn and configure, although for advanced statistical computing this can be a rewarding investment. As the programmer Larry Wall once said: "If ease of use was the highest goal, we'd all be driving golf carts." The `http://admb-project.org/tools/editors/emacs` page contains some pointers for setting up and learning Emacs. There are, however, good reasons why many users may not feel like adopting Emacs as their main editor, but would still appreciate a dedicated environment for ADMB and TMB.

The rest of this tutorial demonstrates how AD Studio can used without learning the details of Emacs. This is achieved with an unusual `.emacs` configuration file that emulates common keybindings of basic editors, while disabling some of the most used Emacs keybindings. This `.emacs` file is therefore not intended for experienced Emacs users, although they may find it an interesting read.

2.2 Software components

AD Model Builder (**ADMB**) and Template Model Builder (**TMB**) are software platforms that utilize automatic differentiation (AD) and Laplace approximation to estimate parameters and random effects in a fast and reliable way. ADMB and TMB generally give the same model fit, but they offer different features. ADMB is a stand-alone application that tends to have better MCMC functionality, while TMB is an R package that has more convenient and faster estimation of random effects.

The **Emacs** editor has dedicated modes for editing different file types and for other tasks. In addition to standard Emacs features, AD Studio includes modes to work with ADMB, LaTeX, Markdown, R, and TMB.

The **GCC** C++ compiler and **GDB** debugger are provided by Rtools. By using the same compiler to build ADMB and TMB models, AD Studio makes it easy to work with ADMB and TMB on the same machine without running into compiler version conflicts.

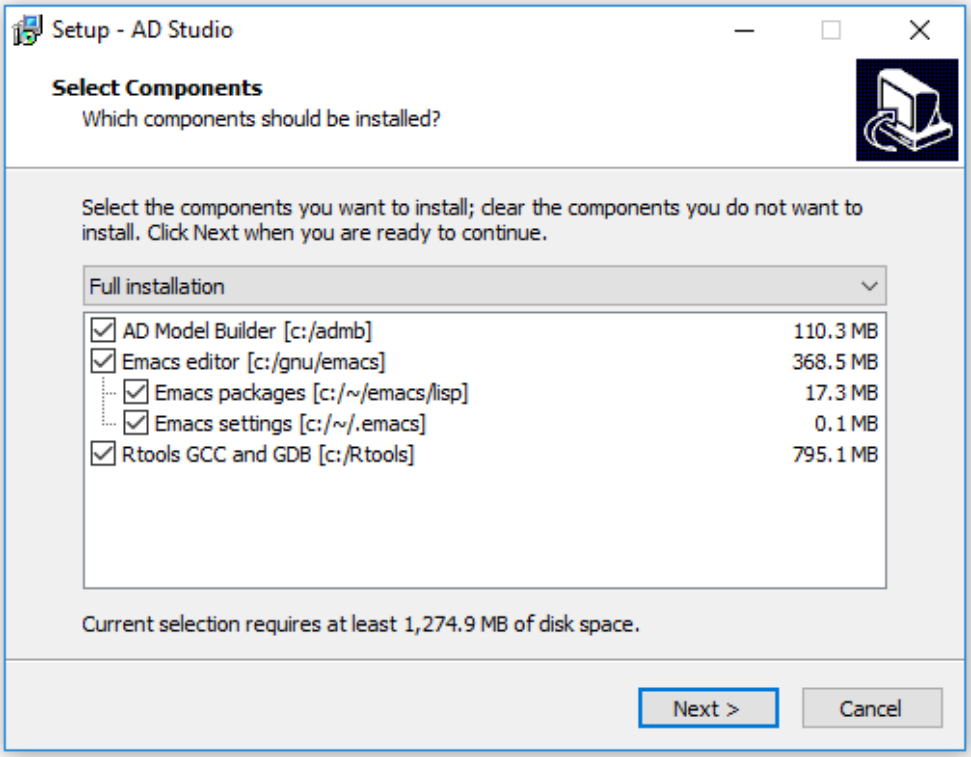
R is required for using TMB, but is not installed as part of AD Studio. Users are expected to install R and the TMB package on their own, as these are frequently updated.

2.3 Installation

AD Studio consists of components that can be set up individually by hand (Windows, Linux, Mac) or using an installer (Windows).

Installer

The Windows installer `adstudio-10.exe` sets up ADMB, GCC, GDB, Emacs, and the Emacs packages, along with file associations and environment variables to glue everything together.



Once installed, the software components reside in four different root directories:

Directory	Component
`c:/~`	Emacs packages and settings
`c:/admb`	ADMB
`c:/gnu`	Emacs
`c:/Rtools`	Rtools

This directory structure can be practical for setting up other free statistical software, such as R. By separating the main program (`c:/gnu/r`) from the user settings (`c:/~/.Rprofile`, `c:/~/Rconsole`) and user libraries (`c:/~/r/library`), the main program can be removed and upgraded without affecting the user setup.

One thing to keep in mind is that the installer modifies the user `PATH` and file associations. In rare cases, users may need to reconfigure these according to taste and needs after installing AD Studio. Advanced users may choose to deselect these options during the installation for this reason.

Manual setup

Windows users can also set up and configure ADMB, Emacs, and Rtools by hand, starting from the `adstudio-10.zip` kit. The following guidelines may be useful for that:

`http://admb-project.org/docs`

`http://admb-project.org/tools/editors/emacs/install`

`http://admb-project.org/tools/editors/emacs/configure`

`https://cran.r-project.org/bin/windows/Rtools/`

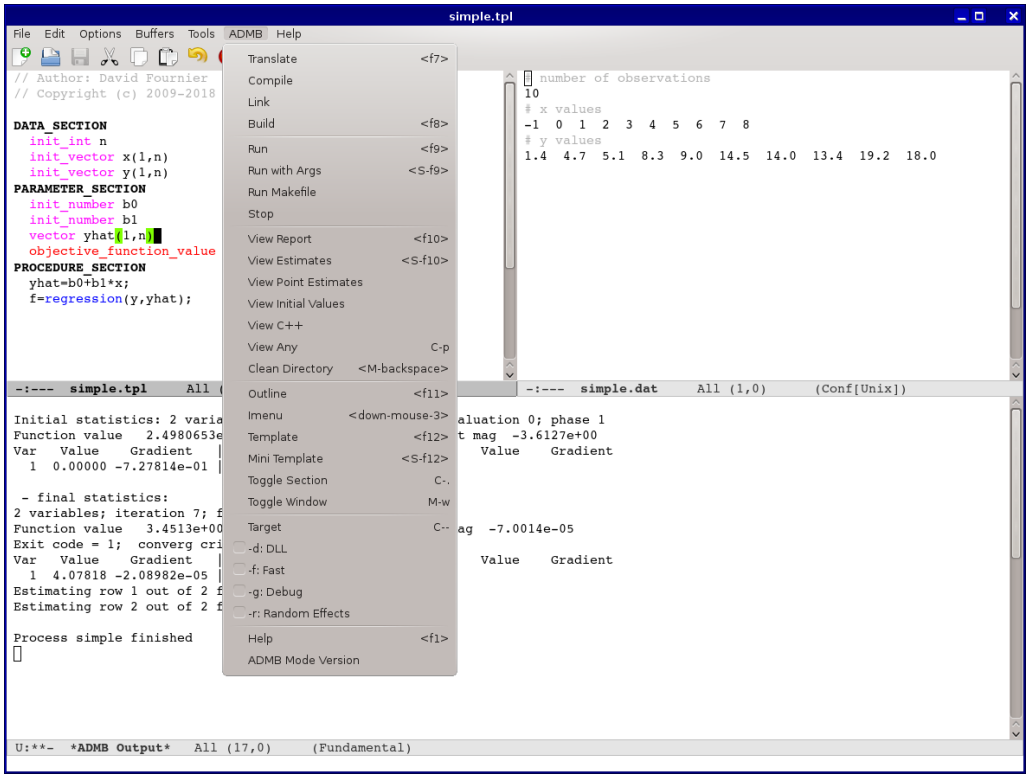
See also the chapters on [Configuration], page 24 and [Troubleshooting], page 25 in this manual.

AD Studio Linux/Mac

Setting up AD Studio for Linux or Mac is equivalent to the "manual setup" described above, so the same guidelines apply. The key steps are:

1. Install ADMB, GNU Emacs, GCC (including the C++ component), and GDB.
2. Download the AD Studio `.emacs` configuration file and place it in `~/.emacs` to apply the simplified Emacs user interface.
3. Download Emacs packages and place inside `~/emacs/lisp` to provide mode-specific syntax highlighting and commands.

The screenshots in the next chapter are from Windows, but AD Studio looks very similar in Linux:



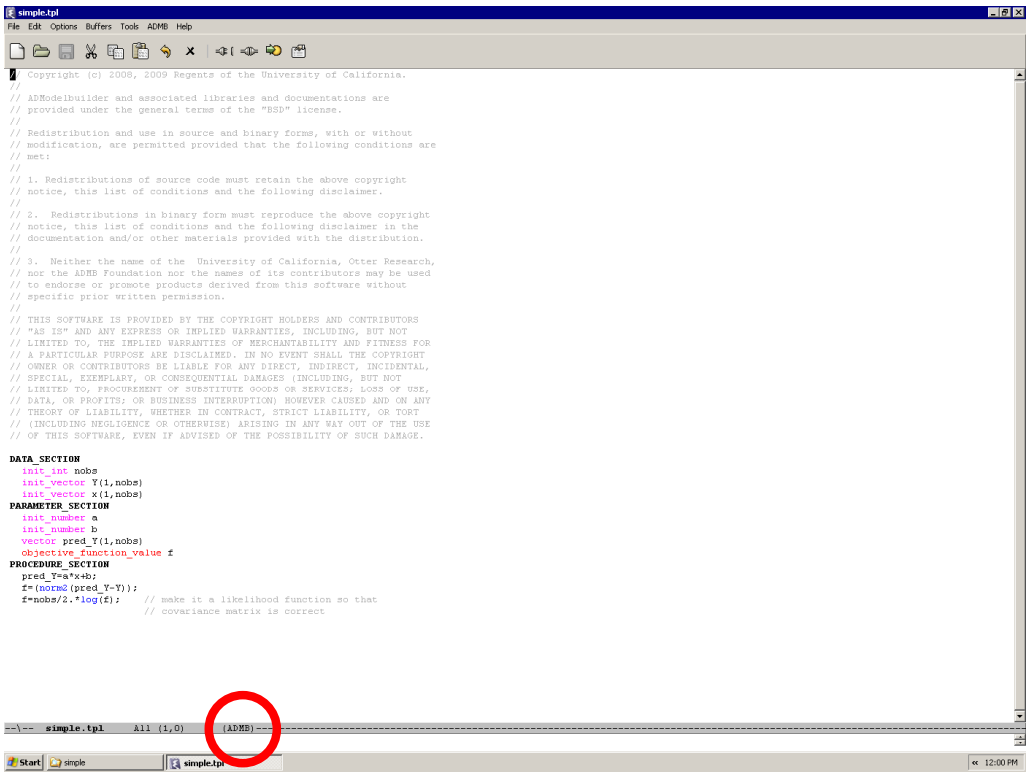
3 ADMB tutorial

3.1 Create a working copy of `simple`

First open Windows Explorer and create a folder called `c:/simple`. Then navigate to `c:/admb/admb120-gcc493-win64/examples/admb/simple` and copy the model and data files, creating:

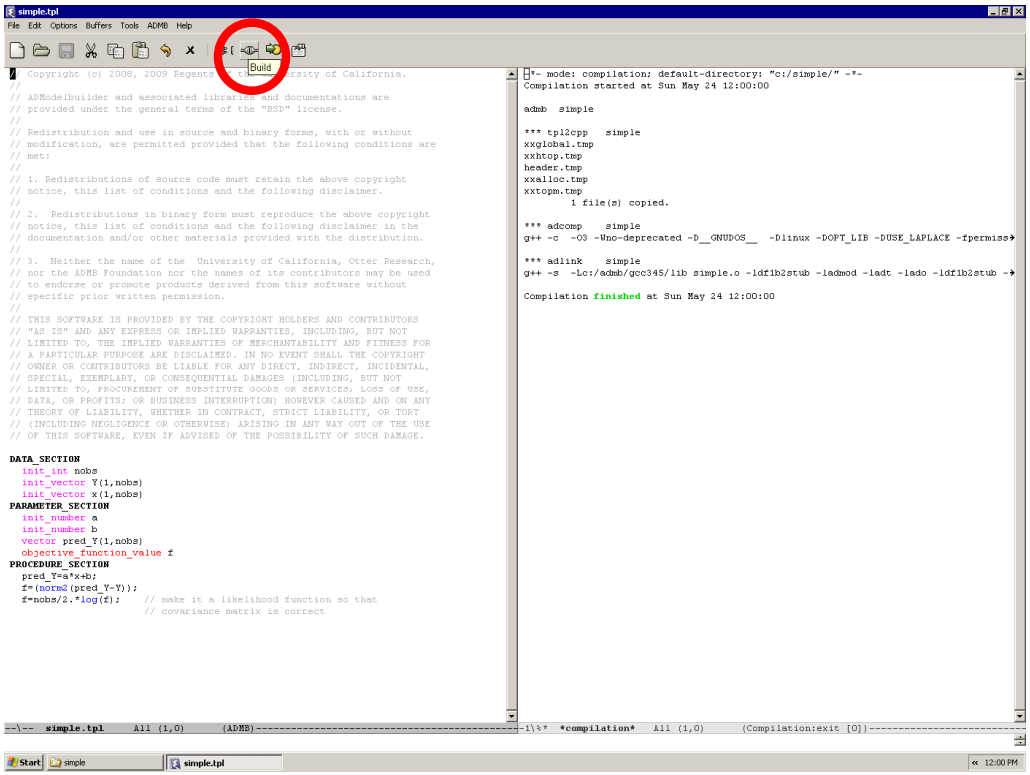
`c:/simple/simple.dat`
`c:/simple/simple.tpl`

Now double-click `simple.tpl` in the `c:/simple` folder. The file should open in Emacs in `admb-mode` (see red circle) and the code should be in color:

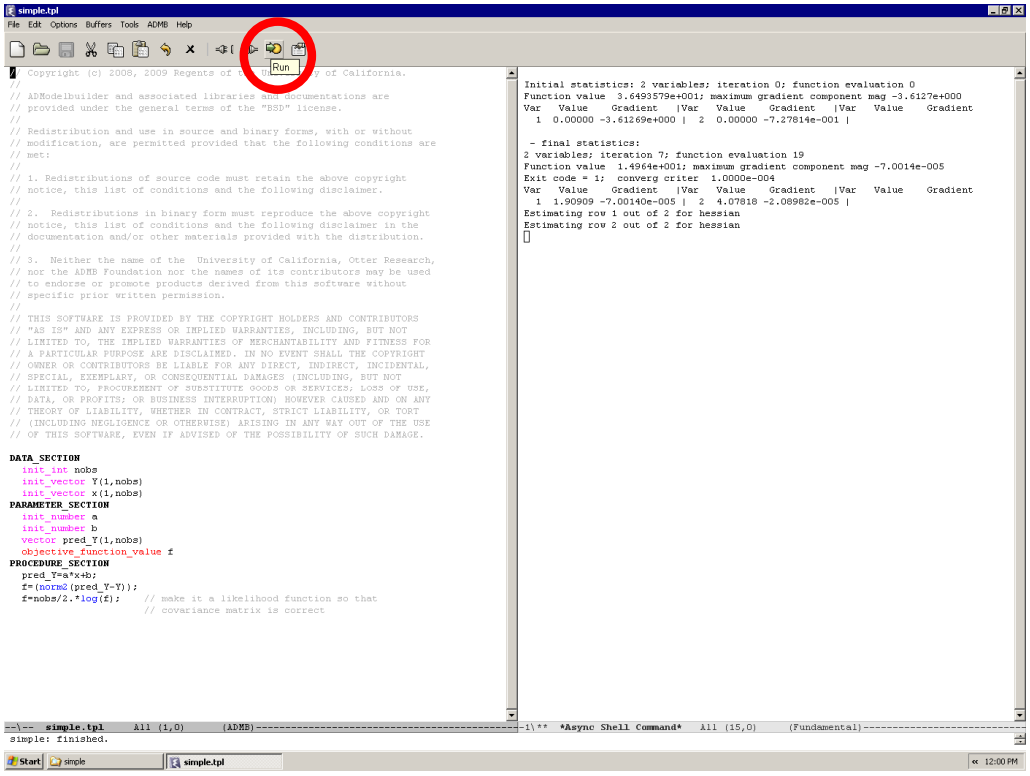


3.2 Build, run, and view the results

Build the model by clicking the icon, or press `f8`:



Run the model by clicking the icon, or press `f9`:

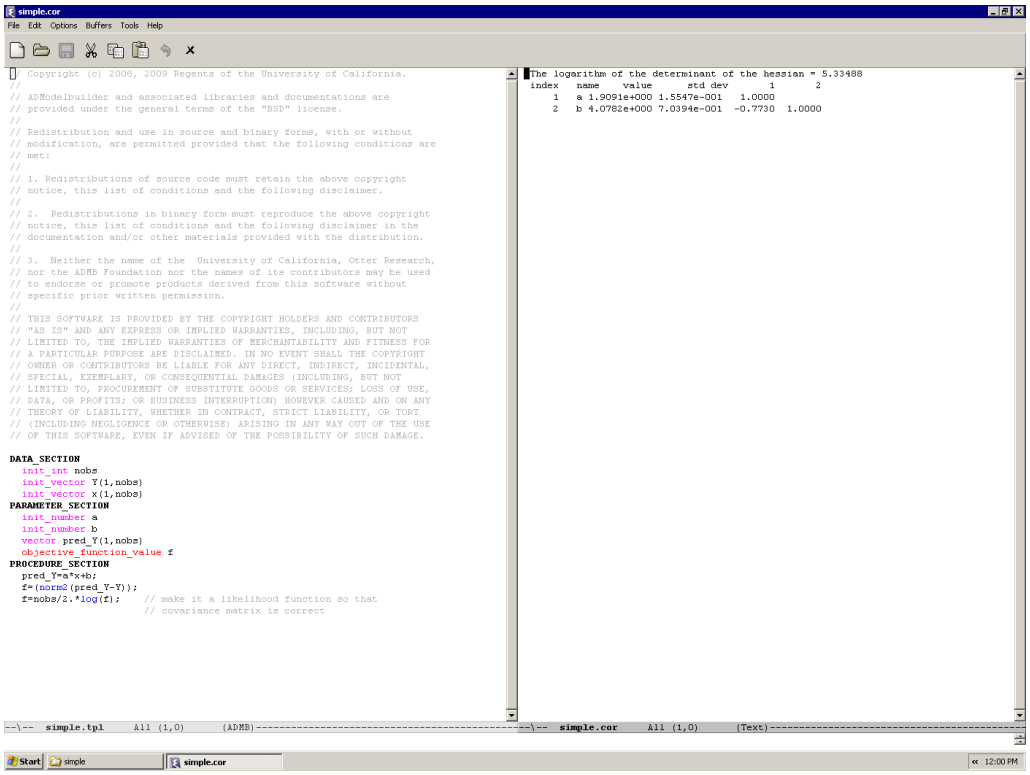


Many ADMB models output their results to a `.rep` report file, and AD Studio provides the icon and `f10` key to open the report file. The `simple` model outputs no report file, but the parameter estimates, standard errors, and correlations are found in the `.cor` file.

This is an opportunity to introduce basic buffer and window management. In Emacs, a buffer is like a page, often representing a file, but sometimes other things, like compilation and command output. The Emacs screen is divided into one or more windows, where each window shows one buffer, while other buffers reside in the background. Explore the ***Buffers*** menu, as well as buffer-related [Keybindings], page 19. A window can be split in two by selecting ***New Window Below*** (`C-x 2`) or ***New Window on Right*** (`C-x 3`) in the ***File*** menu. The `escape` key lets one window fill the Emacs screen.

Try out different ways to open the `.cor` file:

1. Press `escape` to maximize the active window. Then click the icon or press `C-o` (`Ctrl` and `o`) and select `c:/simple/simple.cor`.
2. Press `escape` to maximize the active window, `C-x 3` to split into two windows, and select the window on the right with a mouse click or `f6`. Click the icon or press `C-o` and select `c:/simple/simple.cor`.
3. Click the ***ADMB*** → ***View Estimates*** menu entry or press `S-f10`.
4. Click the ***ADMB*** → ***View Any*** menu entry or press `C-p`, then type 'cor' and `return`.



After viewing, maximize a window by pressing `escape`, or close a window by clicking the `×` icon or pressing `C-w` or `C-f4`.

Note how the *ADMB* menu and toolbar icons are only available when the active window is in `admb-mode`. Press `f2` at any point to switch a window to `admb-mode`.

3.3 Debug

Types of bugs

Bugs in ADMB models can be categorized by the point of discovery:

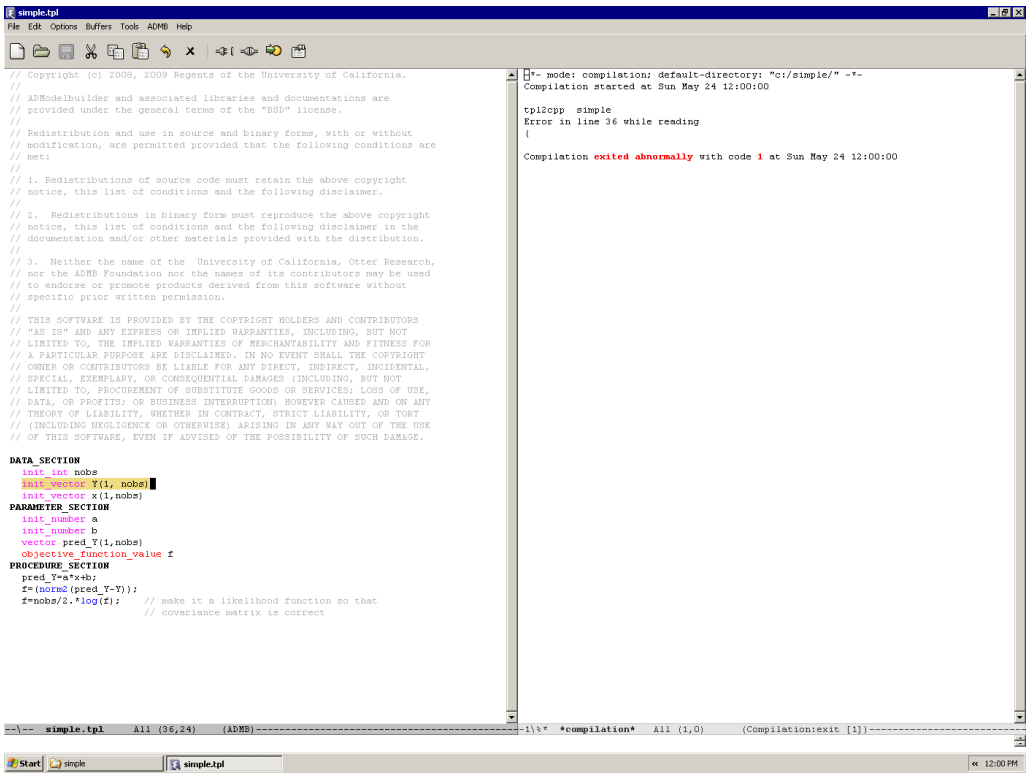
1. `tpl2cpp` reports a bug (cannot translate)
2. `g++` reports a bug (cannot compile or link)
3. The model builds fine, but crashes or writes no output when run (no results)
4. The model runs fine, but not like it is supposed to (strange results)

Locating bugs

- Warnings or error messages indicate line number, or function/variable name
- Insert lines of code that print informative messages during runtime
- Comment out parts of the code
- Use a debugger

Example 1: `tpl2cpp` reports a bug

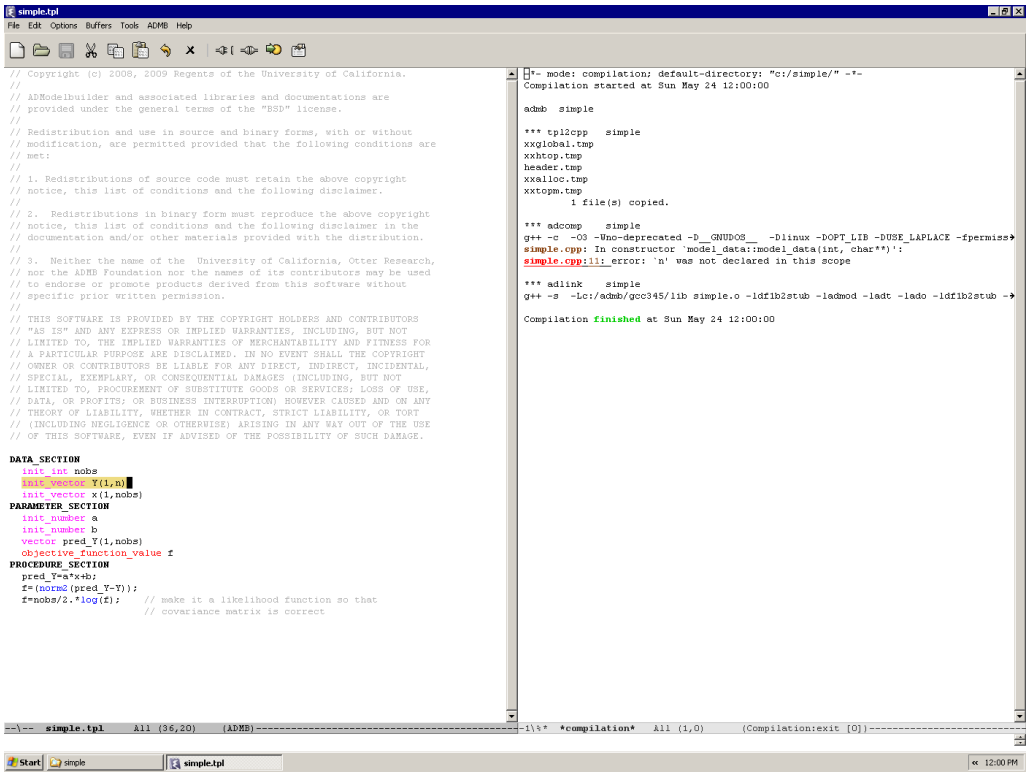
Create a bug by inserting an extra space inside a vector declaration: `init_vector Y(1,nobs)` → `init_vector Y(1, nobs)`. Then click the icon or press `f7` to translate TPL to C++:



The `tpl2cpp` translator reports an error in line 36 of `simple.tpl`. Click *`Edit`* → *`Go To`* → *`Goto Line`* or press `C-g` to move the cursor to that line, and then remove the unwanted space.

Example 2: g++ reports a bug

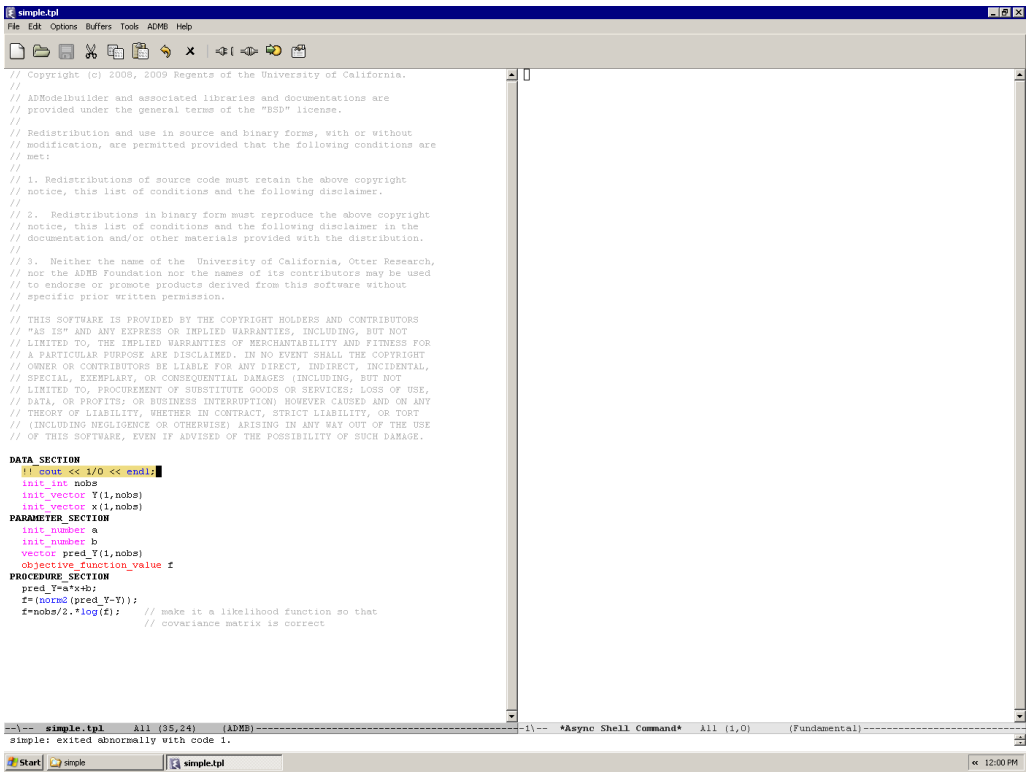
Create a bug by referring to a nonexistent variable: `init_vector Y(1,nobs)` → `init_vector Y(1,n)`. Then click the icon or press `f8` to build the model:



The `g++` compiler reports an error in line 11 of `simple.cpp`. Click the highlighted filename to open the C++ source file with the cursor in that line. After realizing what the problem is (with the help of the error message *`'n' was not declared in this scope`*), go back to the ADMB code in `simple.tpl` and change the '`n`' to '`nobs`'.

Example 3: No results

Create a bug by dividing by zero at the top of the `DATA_SECTION`: `!! cout << 1/0 << endl;`. Then click the icon or press `f8` to build the model. Ignoring the warning, click or press `f9` to run the model:



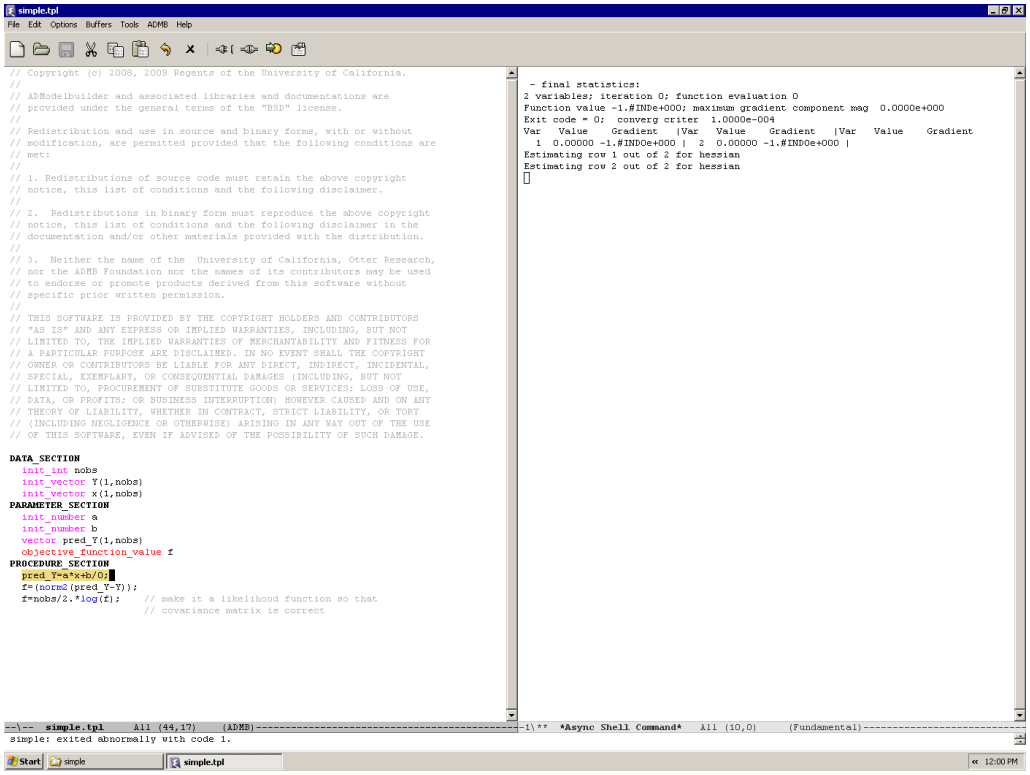
The shell command `simple` exits abnormally with code 1, a generic code for failure. The easiest way to search for this bug is to insert informative messages in the code, like

```
DATA_SECTION
  !! cout << "DATA_SECTION begins" << endl;
  ...
  !! cout << "DATA_SECTION ends" << endl;
```

and/or simplify the model, possibly by commenting out parts of the code. After narrowing the search step by step, the problematic line(s) can be changed or removed. To comment or uncomment large parts of code, use the `M-;` keystroke (see [Keybindings], page 19).

Example 4: Strange results

Create a bug by dividing by zero in a `PROCEDURE_SECTION` assignment: `pred_Y=a*x+b;` → `pred_Y=a*x+b/0;`. Then click the icon or press `f8` to build the model (this time there is no compiler warning). Click or press `f9` to run the model:



The ADMB on-screen report indicates successful convergence (exit code 0) with an objective function value of '`-1.#INDe+000`', while Emacs reports failure (exit code 1). The easiest way to search for this bug is to insert informative messages in the code, like

```
PROCEDURE_SECTION
  cout << "The value of a   is: " << a   << endl;
  cout << "The value of a*x is: " << a*x << endl;
  cout << "The value of b   is: " << b   << endl;
  cout << "The value of b/0 is: " << b/0 << endl;
  cout << "The value of f   is: " << f   << endl;
```

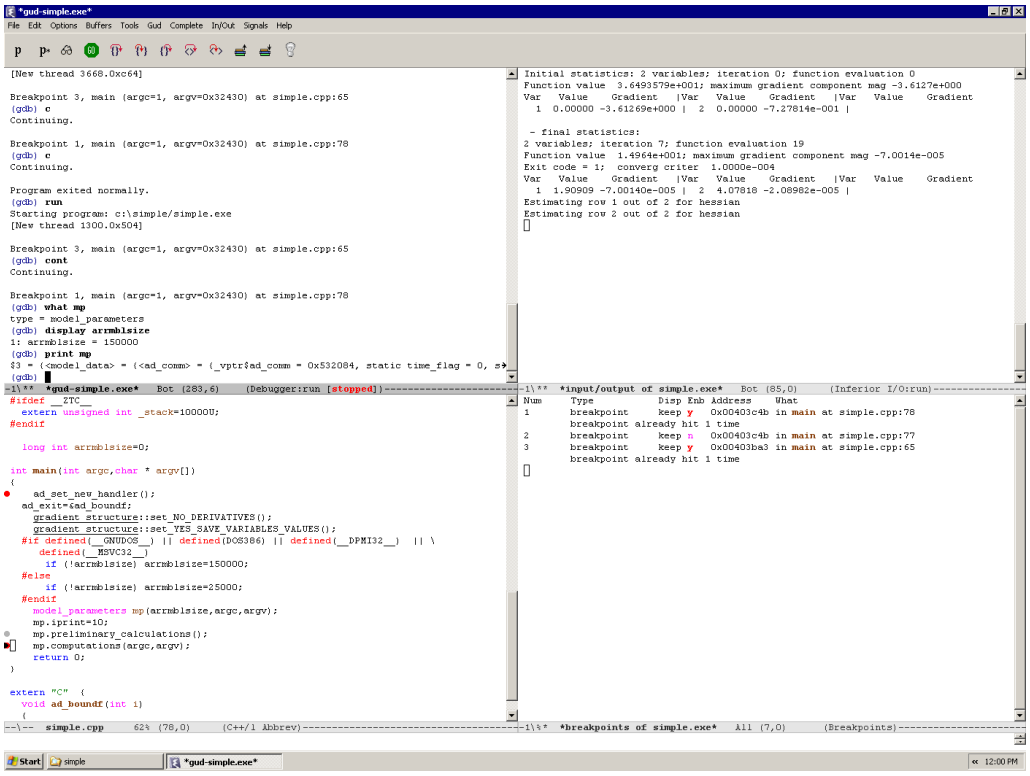
and/or simplify the model, possibly by commenting out parts of the code. A more advanced option is to use a debugger.

GDB: When the going gets tough

GNU Emacs and GCC can interact closely with the GDB debugger—these programs were all created by the same programmer, Richard Stallman. A program must fulfill two conditions before debugging:

- The model executable (e.g., `simple.exe`) must build successfully, so a debugger is only helpful for bugs of type 3 and 4 (see [Types of bugs], page 9).
- The model executable must include debugging symbols. To embed debugging symbols with AD Studio, either select "Debug" compilation from the *ADMB* → *Target* menu, or press `C-- g` (`Ctrl` and `-`, then `g`), followed by `return`.

Using GDB to debug an ADMB model is beyond the scope of this tutorial, but when simpler debugging methods fail, it is time to click *Tools* → *Debugger (GDB)*:



4 TMB tutorial

4.1 Create a working copy of `mini`

Directory

Open AD Studio and create a working directory called `c:/demo`:

```
M-n
c:/demo
C-d
c:/demo
```

The `M-n` keystroke means hold `Alt` and press `n`. This creates a new directory.

The `C-d` keystroke means hold `Ctrl` and press `d`. This sets the current directory.

See [Keybindings], page 19, for a full list of AD Studio keybindings.

Model

Now switch to `tmb-mode` (`f3`) to work with TMB.

AD Studio comes with a built-in TMB example model, which can be used as a minimal template to create a new model. To create a mini template in the current directory, find the menu above the editor window and select *`TMB`* → *`Mini Template`* or press `f12` and `return`.

The screen will now split in two parts, which is the default state of a TMB session in AD Studio. The TMB model code (C++) is in the main window and the compilation script (R) in a secondary window. The cursor remains in the TMB model window, which makes TMB-specific commands available via the menu and keybindings.

4.2 Build, run, and view the results

The compilation script contains R code that compiles and runs the model. There are two alternative ways to build a TMB model:

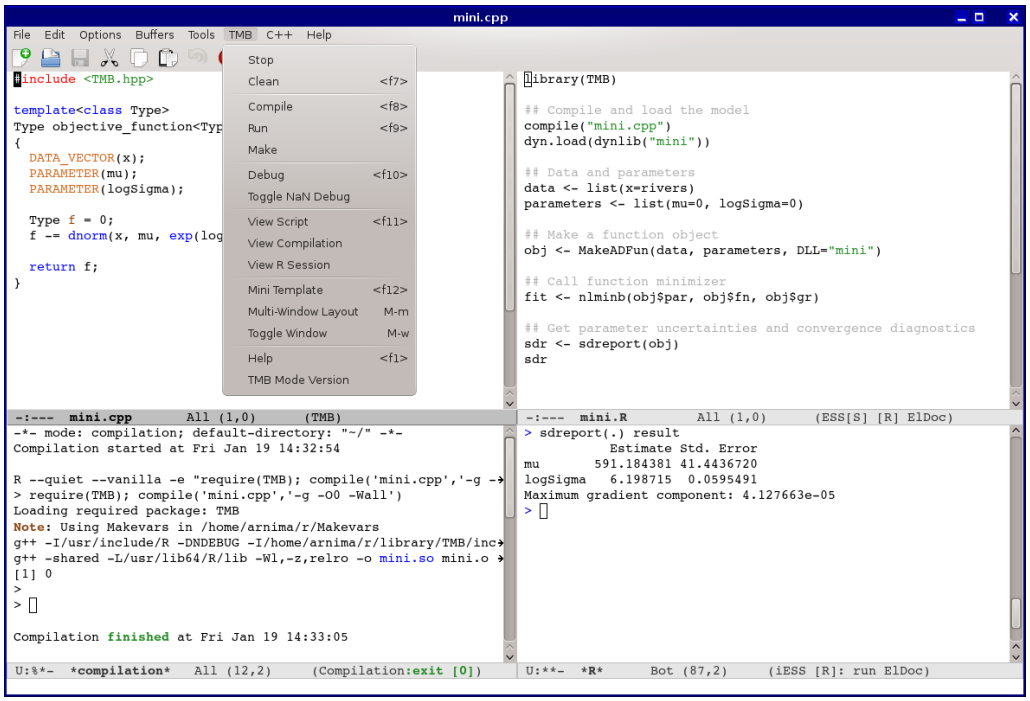
- *`TMB`* → *`Build`* or `f8`: only build the model
- *`TMB`* → *`Run`* or `f9`: run R script, typically builds (if not already built) and runs the model

Manage buffers

The secondary window is now showing either compilation output or an active R session, but the R script is no longer visible. To switch between the R script, compilation output, and the R session, select *`View Script`* (`f11`), *`View Compilation`*, or *`View R Session`* in the TMB menu.

> Note how the *`TMB`* menu is only available when the active window is in `tmb-mode`. Press `f3` at any point to switch a window to `tmb-mode`.

This is a good time to revisit basic buffer and window management. In Emacs, a buffer is like a page, often representing a file, but sometimes other things, like compilation and command output. The Emacs screen is divided into one or more windows, where each window shows one buffer, while other buffers reside in the background. Explore the *`Buffers`* menu, as well as buffer-related [Keybindings], page 19. A window can be split in two by selecting *`New Window Below`* (`C-x 2`) or *`New Window on Right`* (`C-x 3`) in the *`File`* menu. The `escape` key lets one window fill the Emacs screen.



Run R script

While working with TMB models, it is often convenient to keep the cursor inside the TMB window containing the C++ code. This way, the `tmb-mode` facilities (menu, keybindings, etc.) remain active.

Positioning the cursor inside an R script window, however, activates `R-mode` that is part of the Emacs Speaks Statistics (ESS) package. It is a feature-rich mode that supports a variety of R tasks, including package development and debugging, as reflected in the *ESS* menu items. Inside an R script buffer, `f9`, `f10`, and `f11` can be used to run R scripts, a line/paragraph at a time or the entire script. See [R mode keybindings], page 21.

4.3 Debug

To debug a TMB model in AD Studio, select *TMB* → *Debug* or press `f10`. This starts an R session that invokes the GNU Debugger (GDB). To finish the debugging session, type `quit` in the `(gdb)` prompt.

5 User interface

5.1 Menu

ADMB mode

Menu label	*Purpose*	*Emacs command*
Translate	Translate TPL to C++	`admb-tpl2cpp`
Compile	Compile C++ to object code	`admb-compile`
Link	Link object code to exe	`admb-link`
Build	Build executable from TPL	`admb-make`
Run	Run executable	`admb-run`
Run with Args	Run executable with args	`admb-run-args`
Run Makefile	Run Makefile in current dir	`admb-run-makefile`
Stop	Stop current process	`admb-kill-process`
View Report	Open .rep file	`admb-rep`
View Estimates	Open .cor file	`admb-cor`
View Point Estimates	Open .par file	`admb-par`
View Initial Values	Open .pin file	`admb-pin`
View C++	Open C++ file	`admb-cpp`
View Any	Open model file	`admb-open`
Clean Directory	Remove temporary files	`admb-clean`
Outline	Navigate with outline	`admb-outline`
Imenu	Navigate with imenu	`imenu`
Template	Insert template	`admb-template`
Mini Template	Insert minimal template	`admb-template-mini`
Toggle Section	Toggle section indicator	`admb-toggle-section`
Toggle Window	Toggle secondary window	`admb-toggle-window`
Target	Choose what to build	`admb-set-flags`
Help	Show help page	`admb-help`
ADMB Mode Version	Show ADMB Mode version	`admb-mode-version`

TMB mode

Menu label	*Purpose*	*Emacs command*
Stop	Stop current process	`tmb-kill-process`
Clean	Remove *.o, *.so, *.dll	`tmb-clean`
Compile	Build model	`tmb-compile`
Run	Run R script	`tmb-run`
Make	Run makefile	`tmb-make`
Debug	Debug model with GDB	`tmb-debug`
Toggle NaN Debug	Toggle NaN exceptions	`tmb-toggle-nan-debug`
View Script	Show R script	`tmb-open`
View Compilation	Show compilation output	`tmb-show-compilation`
View R Session	Show R sesssion	`tmb-show-r`
Mini Template	Create minimal template	`tmb-template-mini`
Multi-Window Layout	Arrange three windows	`tmb-multi-window`
Toggle Window	Toggle secondary window	`tmb-toggle-window`
Help	Show help page	`tmb-help`
TMB Mode Version	Show TMB Mode version	`tmb-mode-version`

5.2 Toolbar

General

Icon	*Purpose*	*Emacs command*
	New buffer	`new-buffer`
	Open file	`find-file`
	Save file	`save-buffer`
	Cut	`kill-region`
	Copy	`copy-region-as-kill`
	Paste	`cua-paste`
	Undo/redo	`undo`
	Close	`kill-this-buffer`

ADMB mode

Icon	*Purpose*	*Emacs command*
	Translate TPL to C++	`admb-tpl2cpp`
	Build executable from TPL	`admb-make`
	Run executable	`admb-run`
	Open .rep file	`admb-rep`

TMB mode

Icon	*Purpose*	*Emacs command*
	Remove *.o, *.so, *.dll	`tmb-clean`
	Build model	`tmb-compile`
	Run R script	`tmb-run`
	Debug model with GDB	`tmb-debug`

5.3 Keybindings

In combinations, '`S-`' means `Shift`, '`C-`' means `Ctrl`, and '`M-`' means the `Alt` key.

Keystroke	*Purpose*	*Emacs command*
`f1`	Reminder about F2 and F3	`adstudio-help`
`S-f1`	Show AD Studio version	`adstudio-version`
`f2`	ADMB mode	`admb-mode`
`f3`	TMB mode	`tmb-mode`
`f4`	R mode	`R-mode`
`C-f4`	Close	`kill-buffer-maybe-window`
`M-f4`	Quit	`save-buffers-kill-emacs`
`f5`	Reload	`revert-buffer`
`f6`	Other window	`other-window`
`C-f6`	Next buffer	`bs-cycle-next`
`C-,`	Toggle trailing whitespace	`toggle-trailing-whitespace`
`C-.`	Toggle section indicator	`which-function-mode`
`C-a`	Select all	`mark-whole-buffer`
`C-b`	Next buffer	`bs-cycle-next`
`C-c`	Copy	`cua--prefix-override-handler`
`C-d`	Change directory	`cd`
`C-e`	End	`move-end-of-line`
`C-f`	Find, find next	`isearch-forward`
`C-g`	Goto line	`goto-line`
`C-h`	Emacs help system	`help`
`C-l`	Recenter	`recenter`
`C-n`	New file	`new-buffer`
`C-o`	Open	`find-file`
`C-p`	Open in other window	`find-file-other-window`
`C-q`	Quit	`save-buffers-kill-emacs`
`C-r`	Replace	`query-replace`
`C-s`	Save	`save-buffer`
`C-S`	Save as	`write-file`
`C-t`	Toggle F2 and F3 reminder	`adstudio-toggle-reminder`
`C-v`	Paste	`cua-paste`
`C-w`	Close	`kill-buffer-maybe-window`
`C-x`	Cut	`cua--prefix-override-handler`
`C-x 2`	Split window above/below	`split-window-vertically`
`C-x 3`	Split window left/right	`split-window-horizontally`
`C-z`	Undo/redo	`undo`
`C-return`	Rectangle functions	`cua-set-rectangle-mark`
`C-space`	Expand recognized words	`dabbrev-expand`
`M-,`	Delete trailing whitespace	`delete-trailing-spc-tab-m`
`M-;`	Comment/uncomment region	`comment-dwim`
`M-n`	New folder	`make-directory`
`escape`	Cancel dialog, maximize window	`keyboard-escape-quit`

ADMB mode

Keystroke	*Purpose*	*Emacs command*
`f1`	Help for ADMB mode	`admb-help`
`f7`	Translate TPL to C++	`admb-tpl2cpp`
`f8`	Build executable from TPL	`admb-build`
`f9`	Run executable	`admb-run`
`S-f9`	Run executable with args	`admb-run-args`
`f10`	Open .rep file	`admb-rep`
`S-f10`	Open .cor file	`admb-cor`
`f11`	Navigate with outline	`admb-outline`
`S-f11`	Navigate with imenu	`imenu`
`f12`	Insert template	`admb-template`
`S-f12`	Insert minimal template	`admb-template-mini`
`M-backspace`	Remove temporary files	`admb-clean`
`M-up`	Scroll other screen up	`admb-scroll-up`
`M-down`	Scroll other screen down	`admb-scroll-down`
`C--`	Toggle compilation flags	`admb-toggle-flag`
`M-w`	Toggle secondary window	`admb-toggle-window`

TMB mode

Keystroke	*Purpose*	*Emacs command*
`f1`	Help for TMB mode	`tmb-help`
`f7`	Clean	`tmb-clean`
`f8`	Compile	`tmb-compile`
`f9`	Run	`tmb-run`
`f10`	View script	`tmb-open`
`f11`	Debug	`tmb-debug`
`f12`	Mini template	`tmb-template-mini`
`M-up`	Scroll other screen up	`tmb-scroll-up`
`M-down`	Scroll other screen down	`tmb-scroll-down`
`M-m`	Multi-window layout	`tmb-multi-window`
`M-w`	Toggle secondary window	`tmb-toggle-window`

R mode

Keystroke	*Purpose*	*Emacs command*
`f9`	Run line	`ess-eval-region-or-line-and-step`
`f10`	Run paragraph	`ess-eval-region-or-function-or-paragraph-and-step`
`f11`	Run script	`ess-save-buffer-and-eval`

Mouse

Mouse button	*Purpose*	*Emacs command*
`C-left-mouse`	Switch buffers	`mouse-buffer-menu`
`middle-mouse`	Paste	`mouse-yank-primary`
`right-mouse`	Navigate with imenu	`imenu`
`C-wheel-up`	Increase font size	`text-scale-increase`
`C-wheel-down`	Decrease font size	`text-scale-decrease`

6 Hints and tips

6.1 Help pages

Both `admb-mode` and `tmb-mode` come with a help page that is bound to the `f1` key when either of these modes is active. Furthermore, the Emacs help system has a help page for every mode, function and variable, as well as keystrokes:

`C-h m`	Mode
`C-h f`	Function
`C-h v`	Variable
`C-h k`	Keystroke
`C-h a`	Search for command

6.2 Files

The keystrokes `C-n`, `C-o`, `C-s`, and `C-w` can be used to create, open, save, and close files. For convenience, `C-p` opens any file that has a similar name as the ADMB/TMB model, for example, `C-p dat` to open `mymodel.dat`. Directories can be created with `M-n` and switched to with `C-d`. To quit AD Studio, press `C-q`.

6.3 Cut, copy, paste

These three commands are essential for editing any text, and they form a major obstacle for users who try Emacs for the first time.

In almost all other editors, `C-x`, `C-c`, and `C-v` are used to cut, copy, and paste. These are the keybindings that the majority of computer users have memorized and use in a wide variety of applications. The default keybindings in Emacs to cut, copy, and paste (`C-w`, `M-w`, `C-y`) are neither intuitive nor easy to memorize at first. Rebinding keystrokes is easy enough in Emacs, except `C-x` and `C-c` are reserved keystrokes in Emacs, to access many of the most important commands. Rebinding those keystrokes is questionable, in the same way it would be questionable for an R package to redefine things like `q()` or `pi`. There is no perfect solution, but the alternatives include:

1. Bite the bullet and learn the Emacs defaults.
2. Define keybindings to cut/copy/paste that feel convenient, but are not `C-x` or `C-c`.
3. Load special `cua-mode` that overrides the reserved defaults and binds `C-x`, `C-c`, and `C-v` to cut, copy, and paste.

AD Studio takes the third option: CUA mode. Among the drawbacks is that text is not reliably copied between applications, not even between Emacs instances. Workarounds include:

– Copy with `C-insert` and paste with `M-insert`.
– Copy with left mouse button, paste with middle button.

6.4 Undo/redo

The undo command in AD Studio does both undo and redo. When undo is performed repeatedly, it goes further back in the undo history. Any command other than undo will interrupt this sequence, and from that point the previous undo commands become ordinary changes that can be undone, equivalent to redo. Try, for example, copying some text and then paste it three times. Now undo three times, interrupt with a harmless key like the `up` arrow, and then undo again to redo. To undo all changes since last save, it's easiest to reload using the `f5` key.

6.5 Comment/uncomment

The `M-;` key comments or uncomments the highlighted code region, depending on whether the region is already commented or not.

6.6 Secondary window

The default state of an AD Studio session is with the screen split in two parts, with the ADMB/TMB model code (TPL/C++) in the main window and something else in a secondary window. This secondary window can be navigated without leaving the main window, using intuitive keybindings: `M-up`, `M-down`, `M-pgup`, `M-pgdn`, `M-home`, `M-end`.

If the secondary window contains compilation output, `M-up` and `M-down` will navigate between error messages instead of lines.

6.7 Alt key

AD Studio does not emulate exactly the way many editors open menus with the `Alt` key. To open the ***`Edit`*** menu, for example, it is not enough to press `Alt` and `e` simultaneously. AD Studio provides four ways to open the ***`Edit`*** menu:

1. Mouse click on the menu bar
2. Tap `Alt` first and then `e` (Windows)
3. Hold `Alt` and tap `e` twice (Windows)
4. Hold `Alt` and tap `f`, then release `Alt` and tap `right` arrow (Linux)

The idea behind AD Studio, however, is that users can memorize intuitive keystrokes to undo, cut, copy, paste, find, replace, and goto line, without opening the ***`Edit`*** menu. Also don't forget that AD Studio is open source, so users are free to modify any part of the program, including the keybindings defined in the `.emacs` file.

6.8 Vi keybindings

Longtime users of the Vi editor can turn on Emacs `viper-mode`, which is a full-featured Vi emulator for Emacs.

7 Configuration

7.1 Personal `.emacs` file

AD Studio is intended for people who don't know Emacs, are not interested in learning it, and will only use it to work with ADMB. The design goal is that AD Studio should work out of the box and get the job done with minimum fuss.

It is, however, in the nature of modellers to experiment and improve. Users who modify the original `.emacs` file are no longer using AD Studio, but Emacs with packages and a personal `.emacs` file. One reason to modify the `.emacs` file or write a new one from scratch is to install additional Emacs packages. Another reason is to redefine the keybindings, perhaps closer to the Emacs defaults. Other reasons include setting fonts and colors, setting user variables, or defining new user functions. Users with a personal `.emacs` file can update ADMB, GCC, GDB, Emacs and packages independently.

Note that it is not advisable to configure Emacs by clicking *Options* → *Save Options* or *Options* → *Customize Emacs*. Editing the `.emacs` file directly is a more reliable and transparent approach. See http://admb-project.org/tools/editors/emacs for guidelines.

Example

The `f10` key in `admb-mode` runs `admb-rep` to a open a report file:

```
(local-set-key [f10] 'admb-rep) ; menu-bar-open
```

The semicolon starts a comment, reminding that the default behavior of Emacs is to run `menu-bar-open` when `f10` is pressed. In AD Studio, it is easy to activate the menu bar with the mouse or the `Alt` key, so `f10` can be used for something else.

Some users may find it practical to open the report file in an external browser, rather than inside AD Studio. The report file is often best viewed in a large window, and the AD Studio windows are somewhat busy showing other things. It is easy to rebind the `f10` key,

```
(local-set-key [f10] 'admb-rep-browser) ; menu-bar-open
```

but as mentioned in the documentation of `admb-rep-browser`, the `.rep` file ending in Windows may need to be associated with the desired browser program, Firefox or the like.

7.2 Custom startup

It can be practical to have AD Studio available, while using a personal `.emacs` file for most Emacs sessions. For example, an experienced Emacs user may want to test how AD Studio works, or demonstrate it to colleagues, without constantly shuffling `.emacs` files. In Windows, one can place the AD Studio `.emacs` file in `c:/~/adstudio` and then start AD Studio with the shell command:

```
c:/gnu/emacs/bin/runemacs.exe -Q -l c:/~/adstudio/.emacs
```

The `-Q` option tells Emacs to ignore the default startup file(s) and `-l` tells it to load a Lisp file. This command can be used in a start menu or desktop shortcut, with the `c:/~/icons/ad.ico` decorative icon, and similar tricks can be used in Linux and Mac.

8 Troubleshooting

8.1 General usage

The ADMB/TMB menu and toolbar icons disappear

These only appear when the current buffer is in `admb-mode` or `tmb-mode`. Either switch to a source code buffer that is already in `admb-mode` or `tmb-mode`, or press `f2` or `f3` in the current buffer. Other modes may have special menus and toolbar icons that are useful for that mode, see for example the [GDB screenshot], page 13.

The `Tab` key does not indent code properly

ADMB Mode does not know the appropriate indentation of every line, so generally users indent their code manually using `Space` and `Backspace`. The `Tab` key is programmed to insert a number of spaces, as suggested by the previous line, which is sometimes useful.

Lines end with strange `^M` characters

This is how Emacs shows Dos line endings, although in most cases Dos line endings are handled more gracefully. It could be that the file contains mixed line endings (both Dos and Unix), and the simplest solution is to delete all `^M` characters. This can be done using the `M-,` key that deletes all trailing whitespace. It could also be that the Emacs variable `file-name-buffer-file-type-alist` matches the `.tpl` file ending, and the simplest solution is to set that variable to `nil`.

Clicking (*ADMB → Run Makefile*) or (*TMB → Make*) returns an error

Makefiles are a sophisticated build automation tool, not required for general ADMB or TMB usage. This command invokes the `make` program that looks for a file called `Makefile`. If the `make` program or the makefile is not found, an error is returned. This feature is provided for advanced users who have prepared a makefile in the working directory.

8.2 Configuration

Double-clicking a `.tpl` file does not open it in Emacs

The `.tpl` file ending needs to be associated with Emacs. This can be done with registry entries or in Windows Explorer folder options.

Emacs cannot load `admb-mode` or `tmb-mode`

The directory containing the `admb.el` file needs to be in the Emacs variable `load-path`, and the `admb-mode` command needs to be autoloaded in the `.emacs` configuration file. The same holds for `tmb.el` and other Emacs packages.

ADMB compilation commands not recognized

The `PATH` environment variable needs to point to the directories containing the compilation programs (`tpl2cpp`, `tpl2rem`), scripts (`admb`, `adcomp`, `adlink`), and the `g++` program. Likewise, the `ADMB_HOME` environment variable needs to point to the main ADMB directory. Windows environment variables can be set using Dos scripts like `c:/~/bat/admb-set.bat`, or by right-clicking the My Computer icon, then *Properties* → *Advanced* → *Environment Variables* → *User variables* → *New*.

Limited user (i.e., non-administrator) accounts in Windows can also prevent the ADMB-IDE installer from setting the environment variables `ADMB_HOME` and `PATH`. In those cases, the right-click-properties method described above can be used to set the variables after the installation. Many Linux distributions include only the C component of GCC, so users need to install the optional C++ component before using ADMB.

Conflicting compilers and libraries

When developing models using ADMB or TMB, it is important to have only one C++ compiler in the PATH. Likewise, a maximum of one version of ADMB should be in the PATH. Otherwise, errors will occur as objects and libraries of different versions are linked together.

AD Studio uses the C++ compiler that comes with Rtools, which should be used for compiling both ADMB and TMB models. If other C++ compilers are on the machine, it may be necessary to remove them the PATH, either by modifying the PATH environment variable, or temporarily renaming directories.

9 References

ADMB

Fournier, D.A., H.J. Skaug, J. Ancheta, J. Ianelli, A. Magnusson, M.N. Maunder, A. Nielsen, and J. Sibert. 2012. *AD Model Builder: Using automatic differentiation for statistical inference of highly parameterized complex nonlinear models.* Optimization Methods and Software 27:233–249.
http://dx.doi.org/10.1080/10556788.2011.597854

Fournier, D. *An introduction to AD Model Builder for use in nonlinear modeling and statistics.*
http://admb-project.org/docs/manuals/

Skaug, H. and D. Fournier. *Random effects in AD Model Builder: ADMB-RE user guide.*
http://admb-project.org/docs/manuals/

Fournier, D. *AUTODIF: A C++ array language extension with automatic differentiation for use in nonlinear modeling and statistics.*
http://admb-project.org/docs/manuals/

Magnusson, A. 2009. *ADMB-IDE: Easy and efficient user interface.* ADMB Foundation Newsletter 1(3):1–2.
http://admb-foundation.org/wp-content/uploads/Newsletter/ADMBNewsletterJuly2009.pdf

TMB

Kristensen, K., A. Nielsen, C.W. Berg, H. Skaug, and B. Bell. 2016. *TMB: Automatic differentiation and Laplace approximation.* J. Stat. Softw. 70(5):1–21.
http://dx.doi.org/10.18637/jss.v070.i05

Kristensen, K. with contributions by B. Bell, H. Skaug, A. Magnusson, C. Berg, A. Nielsen, M. Maechler, T. Michelot, M. Brooks, A. Forrence, C.M. Albertsen, and C. Monnahan. *Template Model Builder: A general random effect tool inspired by ADMB.*
https://cran.r-project.org/package=TMB

Magnusson, A. 2015. TMB-IDE: Emacs `tmb-mode` without the Emacs.
ftp://ftp.hafro.is/pub/tmb/tmb-ide.pdf

Emacs

Stallman, R. *GNU Emacs manual.*
http://www.gnu.org/software/emacs/manual/emacs.html

Lewis, B., D. LaLiberte, R. Stallman, and the GNU Manual Group. *GNU Emacs Lisp reference manual for Emacs.*
http://www.gnu.org/software/emacs/manual/elisp.html

Chassell, R. *An introduction to programming in Emacs Lisp.*
http://www.gnu.org/software/emacs/manual/eintr.html

Rossini, A.J., R.M. Heiberger, K. Hornik, M. Maechler, R.A. Sparapani, S.J. Eglen, S.P. Luque, H. Redestig, V. Spinu, and L. Henry. *ESS: Emacs Speaks Statistics.*
http://ess.r-project.org/index.php?Section=documentation&subSection=manuals

GCC & GDB

Stallman, R.M. and GCC Developer Community. *Using the GNU Compiler Collection.* http://gcc.gnu.org/onlinedocs/

Stallman, R., R. Pesch, S. Shebs, et al. *Debugging with GDB: The GNU source-level debugger.* http://sourceware.org/gdb/download/onlinedocs/