

Use R!

Series Editors:

Robert Gentleman Kurt Hornik Giovanni G. Parmigiani

For further volumes:

<http://www.springer.com/series/6991>

Dirk Eddelbuettel

Seamless R and C++ Integration with Rcpp

 Springer

Dirk Eddelbuettel
River Forest
Illinois, USA

ISBN 978-1-4614-6867-7 ISBN 978-1-4614-6868-4 (eBook)
DOI 10.1007/978-1-4614-6868-4
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013933242

© The Author 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Lisa, Anna and Julia

Preface

Rcpp is an R add-on package which facilitates extending R with C++ functions.

It is being used for anything from small and quickly constructed add-on functions written either to fluidly experiment with something new or to accelerate computing by replacing an R function with its C++ equivalent to large-scale bindings for existing libraries, or as a building block in entirely new research computing environments.

While still relatively new as a project, **Rcpp** has already become widely deployed among users and developers in the R community. **Rcpp** is now the most popular language extension for the R system and used by over 100 CRAN packages as well as ten BioConductor packages.

This book aims to provide a solid introduction to **Rcpp**.

Target Audience

This book is for R users who would like to extend R with C++ code. Some familiarity with R is certainly helpful; a number of other books can provide refreshers or specific introductions. C++ knowledge is also helpful, though not strictly required. An appendix provides a very brief introduction for C++ to those familiar only with the R language.

The book should also be helpful to those coming to R with more of a C++ programming background. However, additional background reading may be required to obtain a firmer grounding in R itself. Chambers (2008) is a good introduction to the philosophy behind the R system and a helpful source in order to acquire a deeper understanding.

There may also be some readers who would like to see how **Rcpp** works internally. Covering that aspect, however, requires a fairly substantial C++ content and is not what this book is trying to provide. The focus of this book is clearly on how to use **Rcpp**.

Historical Context

Rcpp first appeared in 2005 as a (fairly small when compared to its current size) contribution by Dominick Samperi to the **RQuantLib** package started by Eddelbuettel in 2002 (Eddelbuettel and Nguyen 2012). **Rcpp** became a CRAN package in its own name in early 2006. Several releases (all provided by Samperi) followed in quick succession under the name **Rcpp**. The package was then renamed to **RcppTemplate**; several more releases followed during 2006 under the new name. However, no new releases were made during 2007, 2008, or most of 2009. Following a few updates in late 2009, the **RcppTemplate** package has since been archived on CRAN for lack of active maintenance.

Given the continued use of the package, Eddelbuettel decided to revitalize it. New releases, using the original name **Rcpp**, started in November 2008. These included an improved build and distribution process, additional documentation, and new functionality—while retaining the existing “classic **Rcpp**” interface. While not described here, this API will continue to be provided and supported via the **RcppClassic** package (Eddelbuettel and François 2012c).

Reflecting evolving C++ coding standards (see Meyers 2005), Eddelbuettel and François started a significant redesign of the code base in 2009. This added numerous new features, many of which are described in the package via different vignettes. This redesigned version of **Rcpp** (Eddelbuettel and François 2012a) has become widely used with over ninety CRAN packages depending on it as of November 2012. It is also the version described in this book.

Rcpp continues to be under active development, and extensions are being added. The content described here shall remain valid and supported.

Related Work

Integration of C++ and R has been addressed by several authors; the earliest published reference is probably Bates and DebRoy (2001). The “Writing R Extensions” manual (R Development Core Team 2012d) has also been mentioning C++ and R integration since around that time. An unpublished paper by Java et al. (2007) expresses several ideas that are close to some of our approaches, though not yet fully fleshed out. The **Rserve** package (Urbanek 2003, 2012) acts as a socket server for R. On the server side, **Rserve** translates R data structures into a binary serialization format and uses TCP/IP for transfer. On the client side, objects are reconstructed as instances of Java or C++ classes that emulate the structure of R objects.

The packages **rcppbind** (Liang 2008), **RAbstraction** (Armstrong 2009a), and **RObjects** (Armstrong 2009b) are all implemented using C++ templates. None of them have matured to the point of a CRAN release. **CXXR** (Runnalls 2009) approaches this topic from the other direction: its aim is to completely refactor R on a stronger C++ foundation. **CXXR** is therefore concerned with all aspects of the R interpreter, read-eval-print loop (REPL), and threading; object interchange be-

tween R and C++ is but one part. A similar approach is discussed by Temple Lang (2009a) who suggests making low-level internals extensible by package developers in order to facilitate extending R. Temple Lang (2009b), using compiler output for references on the code in order to add bindings and wrappers, offers a slightly different angle. Lastly, the **rdyncall** package (Adler 2012) provides a direct interface from R into C language APIs. This can be of interest if R programmers want to access lower-level programming interfaces directly. However, it does not aim for the same object-level interchange that is possible via C++ interfaces, and which we focus on with **Rcpp**.

Typographic Convention

The typesetting follows the usage exemplified both by the publisher, and by the *Journal of Statistical Software*. We use

- Sans-serif for programming language such as R or C++
- Boldface for (CRAN or other) software packages such as **Rcpp** or **inline**
- Courier for short segments of code or variables such as `x <- y + z`

We make use of a specific environment for the short pieces of source code interwoven with the main text.

River Forest, IL, USA

Dirk Eddelbuettel

Acknowledgements

Rcpp is the work of many contributors, and a few words of thanks are in order.

Dominick Samperi contributed the original code which, while much more limited in scope than the current **Rcpp**, pointed clearly in the right direction of using C++ templates to convert between R and C++ types.

Romain François has shown impeccable taste in designing and implementing very large parts of **Rcpp** as it is today. The power of the current design owes a lot to his work and boundless energy. Key components such as modules and sugar, as well as lot a of template “magic,” are his contributions. This started as an aside to make object interchange easier for our **RProtoBuf** package—and it has taken us down a completely different, but very exciting road. It has been a pleasure to work with Romain, I remain in awe of his work, and I look forward to many more advances with **Rcpp**.

Doug Bates has been a help from the very beginning: had it not been for some simple macros to pick list components out of *SEXP* types, I may never have started **RQuantLib** a decade ago. Doug later joined this project and has been instrumental in a few key decisions regarding **Rcpp** and **RcppArmadillo** and has taken charge of the **RcppEigen** project.

John Chambers became a key supporter right when *Rcpp modules* started and contributed several important pieces at the gory intersection between R and C++. It was very flattering for Romain and me to hear from John how **Rcpp** is so close to an original design vision of a whole-object interchange between systems which was already present on a hand-drawn Bell Labs designs from the 1970s.

JJ Allaire has become a very important contributor to **Rcpp** and a key supporter of the same idea of an almost natural pairing between R and C++. The *Rcpp attributes* which he contributed are showing a lot of promise, and we expect great things to be built on top of this.

Several other members of the R Core team—notably Kurt Hornik, Uwe Ligges, Martyn Plummer, Brian Ripley, Luke Tierney, and Simon Urbanek—have helped at various points with anything from build issues and portability to finer points of R internals. Last but not least, there would of course be no **Rcpp** if there was no R system to build upon and to extend.

Finally, many members of the R and **Rcpp** communities have been very supportive at different workshops, conference presentations, and via the mailing lists. Numerous good questions and suggestions have come this way. And, of course, it is seeing this work being used so actively which motivates us and keeps us moving forward with **Rcpp**.

Contents

Part I Introduction

1	A Gentle Introduction to Rcpp	3
1.1	Background: From R to C++	3
1.2	A First Example	7
1.2.1	Problem Setting	7
1.2.2	A First R Solution	7
1.2.3	A First C++ Solution	8
1.2.4	Using Inline	9
1.2.5	Using Rcpp Attributes	11
1.2.6	A Second R Solution	12
1.2.7	A Second C++ Solution	12
1.2.8	A Third R Solution	14
1.2.9	A Third C++ Solution	14
1.3	A Second Example	15
1.3.1	Problem Setting	15
1.3.2	R Solution	15
1.3.3	C++ Solution	16
1.3.4	Comparison	17
1.4	Summary	18
2	Tools and Setup	19
2.1	Overall Setup	19
2.2	Compilers	20
2.2.1	General Setup	20
2.2.2	Platform-Specific Notes	21
2.3	The R Application Programming Interface	22
2.4	A First Compilation with Rcpp	23
2.5	The Inline Package	25
2.5.1	Overview	25
2.5.2	Using Includes	27

2.5.3	Using Plugins	29
2.5.4	Creating Plugins	30
2.6	Rcpp Attributes	31
2.7	Exception Handling	32

Part II Core Data Types

3	Data Structures: Part One	39
3.1	The RObject Class	39
3.2	The IntegerVector Class	41
3.2.1	A First Example: Returning Perfect Numbers	42
3.2.2	A Second Example: Using Inputs	43
3.2.3	A Third Example: Using Wrong Inputs	44
3.3	The NumericVector Class	45
3.3.1	A First Example: Using Two Inputs	45
3.3.2	A Second Example: Introducing clone	46
3.3.3	A Third Example: Matrices	47
3.4	Other Vector Classes	48
3.4.1	LogicalVector	48
3.4.2	CharacterVector	49
3.4.3	RawVector	49
4	Data Structures: Part Two	51
4.1	The Named Class	51
4.2	The List aka GenericVector Class	52
4.2.1	List to Retrieve Parameters from R	53
4.2.2	List to Return Parameters to R	54
4.3	The DataFrame Class	55
4.4	The Function Class	56
4.4.1	A First Example: Using a Supplied Function	56
4.4.2	A Second Example: Accessing an R Function	56
4.5	The Environment Class	57
4.6	The S4 Class	58
4.7	ReferenceClasses	59
4.8	The R Mathematics Library Functions	60

Part III Advanced Topics

5	Using Rcpp in Your Package	65
5.1	Introduction	65
5.2	Using Rcpp.package.skeleton	66
5.2.1	Overview	66
5.2.2	R Code	67
5.2.3	C++ Code	68
5.2.4	DESCRIPTION	69
5.2.5	Makevars and Makevars.win	69

- 5.2.6 NAMESPACE 71
- 5.2.7 Help Files 71
- 5.3 Case Study: The **wordcloud** Package 73
- 5.4 Further Examples 74
- 6 Extending Rcpp** 75
 - 6.1 Introduction 75
 - 6.2 Extending Rcpp::wrap 76
 - 6.2.1 Intrusive Extension 76
 - 6.2.2 Nonintrusive Extension 77
 - 6.2.3 Templates and Partial Specialization 78
 - 6.3 Extending Rcpp::as 78
 - 6.3.1 Intrusive Extension 78
 - 6.3.2 Nonintrusive Extension 79
 - 6.3.3 Templates and Partial Specialization 79
 - 6.4 Case Study: The **RcppBDT** Package 80
 - 6.5 Further Examples 82
- 7 Modules** 83
 - 7.1 Motivation 83
 - 7.1.1 Exposing Functions Using **Rcpp** 83
 - 7.1.2 Exposing Classes Using Rcpp 84
 - 7.2 Rcpp Modules 86
 - 7.2.1 Exposing C++ Functions Using Rcpp Modules 86
 - 7.2.2 Exposing C++ Classes Using Rcpp Modules 90
 - 7.3 Using Modules in Other Packages 98
 - 7.3.1 Namespace Import/Export 98
 - 7.3.2 Support for Modules in Skeleton Generator 99
 - 7.3.3 Module Documentation 100
 - 7.4 Case Study: The **RcppCNPY** Package 100
 - 7.5 Further Examples 102
- 8 Sugar** 103
 - 8.1 Motivation 103
 - 8.2 Operators 105
 - 8.2.1 Binary Arithmetic Operators 105
 - 8.2.2 Binary Logical Operators 106
 - 8.2.3 Unary Operators 106
 - 8.3 Functions 107
 - 8.3.1 Functions Producing a Single Logical Result 107
 - 8.3.2 Functions Producing Sugar Expressions 107
 - 8.3.3 Mathematical Functions 113
 - 8.3.4 The d/q/p/q Statistical Functions 114
 - 8.4 Performance 115
 - 8.5 Implementation 116

8.5.1	The Curiously Recurring Template Pattern	117
8.5.2	The VectorBase Class	117
8.5.3	Example: sapply	118
8.6	Case Study: Computing π Using <i>Rcpp sugar</i>	122

Part IV Applications

9	RInside	127
9.1	Motivation	127
9.2	A First Example: Hello, World!	128
9.3	A Second Example: Data Transfer	131
9.4	A Third Example: Evaluating R Expressions	132
9.5	A Fourth Example: Plotting from C++ via R	133
9.6	A Fifth Example: Using RInside Inside MPI	134
9.7	Other Examples	135
10	RcppArmadillo	139
10.1	Overview	139
10.2	Motivation: FastLm	140
10.2.1	Implementation	140
10.2.2	Performance Comparison	142
10.2.3	A Caveat	144
10.3	Case Study: Kalman Filter Using RcppArmadillo	146
10.4	RcppArmadillo and Armadillo Differences	152
11	RcppGSL	155
11.1	Introduction	155
11.2	Motivation: FastLm	156
11.3	Vectors	158
11.3.1	GSL Vectors	158
11.3.2	RcppGSL::vector	159
11.3.3	Mapping	161
11.3.4	Vector Views	161
11.4	Matrices	163
11.4.1	Creating Matrices	163
11.4.2	Implicit Conversion	163
11.4.3	Indexing	163
11.4.4	Methods	164
11.4.5	Matrix Views	164
11.5	Using RcppGSL in Your Package	164
11.5.1	The <code>configure</code> Script	165
11.5.2	The <code>src</code> Directory	166
11.5.3	The <code>R</code> Directory	167
11.6	Using RcppGSL with inline	168
11.7	Case Study: GSL -Based B-Spline Fit Using RcppGSL	169

- 12 RcppEigen** 177
 - 12.1 Introduction 177
 - 12.2 Eigen classes 178
 - 12.2.1 Fixed-Size Vectors and Matrices 178
 - 12.2.2 Dynamic-Size Vectors and Matrices 179
 - 12.2.3 Arrays for Per-Component Operations 180
 - 12.2.4 Mapped Vectors and Matrices and Special Matrices 181
 - 12.3 Case Study: Kalman filter using RcppEigen 182
 - 12.4 Linear Algebra and Matrix Decompositions 183
 - 12.4.1 Basic Solvers 183
 - 12.4.2 Eigenvalues and Eigenvectors 184
 - 12.4.3 Least-Squares Solvers 185
 - 12.4.4 Rank-Revealing Decompositions 185
 - 12.5 Case Study: C++ Factory for Linear Models in **RcppEigen** 186

Part V Appendix

- A C++ for R Programmers** 195
 - A.1 Compiled Not Interpreted 195
 - A.2 Statically Typed 197
 - A.3 A Better C 198
 - A.4 Object-Oriented (But Not Like S3 or S4) 200
 - A.5 Generic Programming and the STL 201
 - A.6 Template Programming 203
 - A.7 Further Reading on C++ 204

References 207

Subject Index 211

Software Index 217

Author Index 219

List of Tables

Table 1.1	Run-time performance of the recursive Fibonacci examples . .	10
Table 1.2	Run-time performance of the different VAR simulation implementations	17
Table 8.1	Run-time performance of <i>Rcpp sugar</i> compared to R and manually optimized C++	116
Table 8.2	Run-time performance of <i>Rcpp sugar</i> compared to R for simulating π	124
Table 11.1	Correspondence between GSL vector types and templates defined in RcppGSL	161
Table 11.2	Correspondence between GSL vector view types and templates defined in RcppGSL	162
Table 12.1	Mapping between Eigen matrix and vector types, and corresponding array types	181
Table 12.2	lmBenchmark results for the RcppEigen example	191

List of Figures

Figure 1.1	Plotting a density in R	5
Figure 1.2	Plotting a density and bootstrapped confidence interval in R	6
Figure 1.3	Fibonacci spiral based on first 34 Fibonacci numbers	8
Figure 9.1	Combining RInside with the Qt toolkit for a GUI application	135
Figure 9.2	Combining RInside with the Wt toolkit for a web application	136
Figure 10.1	Object trajectory and Kalman filter estimate	149
Figure 11.1	Artificial data and B-spline fit	175

List of Listings

1.1	Plotting a density in R	4
1.2	Plotting a density and bootstrapped confidence interval in R	4
1.3	Fibonacci number in R via recursion	7
1.4	Fibonacci number in C++ via recursion	8
1.5	Fibonacci wrapper in C++	9
1.6	Fibonacci number in C++ via recursion, using inline	9
1.7	Fibonacci number in C++ via recursion, using Rcpp attributes	11
1.8	Fibonacci number in C++ via recursion, via Rcpp attributes and <code>sourceCpp</code>	11
1.9	Fibonacci number in R via memoization	12
1.10	Fibonacci number in C++ via memoization	12
1.11	Fibonacci number in R via iteration	14
1.12	Fibonacci number in C++ via iteration	14
1.13	VAR(1) of order 2 generation in R	16
1.14	VAR(1) of order 2 generation in C++	16
1.15	Comparison of VAR(1) run-time between R and C++	17
2.1	A first manual compilation with Rcpp	23
2.2	A first manual compilation with Rcpp using Rscript	24
2.3	Using the first manual compilation from R	24
2.4	Convolution example using inline	26
2.6	Using inline with <code>include=</code>	27
2.5	Program source from convolution example using inline in verbose mode	28
2.7	A first RcppArmadillo example for inline	29
2.8	Creating a plugin for use with inline	30
2.9	Example of new <code>cppFunction</code>	31
2.10	Example of new <code>cppFunction</code> with plugin	32
2.11	C++ example of throwing and catching an exception	32
2.12	Using C++ example of throwing and catching an exception	33
2.13	C++ example of example from Rcpp-type checks	33
2.14	C++ macros for Rcpp exception handling	34

2.15	inline version of C++ example of throwing and catching an exception	34
2.16	Rcpp attributes version of C++ example of throwing and catching an exception	34
3.1	A function to return four perfect numbers	42
3.2	A function to reimplement <code>prod()</code>	43
3.3	A second function to reimplement <code>prod()</code>	43
3.4	Testing the <code>prod()</code> function with floating-point inputs	44
3.5	Testing the <code>prod()</code> function with inappropriate inputs	45
3.6	A function to return a generalized sum of powers	45
3.7	Declaring two vectors from the same <code>SEXP</code> type	46
3.8	Declaring two vectors from the same <code>SEXP</code> type using <code>clone</code>	46
3.9	Using Rcpp sugar to compute a second vector	47
3.10	Declaring a three-dimensional vector	47
3.11	A function to take square roots of matrix elements	48
3.12	A function to assign a logical vector	48
3.13	A function to assign a character vector	49
4.1	A named vector in R	51
4.2	A named vector in C++	52
4.3	A named vector in C++, second approach	52
4.4	Using the <code>List</code> class for parameters	53
4.5	Using a <code>List</code> to return objects to R	54
4.6	Using the <code>DataFrame</code> class	55
4.7	Using a <code>Function</code> passed as argument	56
4.8	Using a <code>Function</code> accessed from R	57
4.9	Using a <code>Function</code> via an <code>Environment</code>	57
4.10	Assigning in the global environment	58
4.11	A simple example for accessing S4 class elements	58
4.12	A simple example for accessing S4 class elements	59
4.13	Example use of <code>Rmath.h</code> functions	60
5.1	A first <code>Rcpp.package.skeleton</code> example	66
5.2	Files created by <code>Rcpp.package.skeleton</code>	67
5.3	R function <code>rcpp_hello_world</code>	67
5.4	C++ header file <code>rcpp_hello_world.h</code>	68
5.5	C++ source file <code>rcpp_hello_world.cpp</code>	68
5.6	Calling R function <code>rcpp_hello_world</code>	69
5.7	DESCRIPTION file for skeleton package	69
5.8	Makevars file for skeleton package	70
5.9	Makevars.win file for skeleton package	70
5.10	NAMESPACE file for skeleton package	71
5.11	Manual page <code>mypackage-package.Rd</code> for skeleton package	71
5.12	Manual page <code>rcpp_hello_world.Rd</code> for skeleton package	72
5.13	Function <code>is_overlap</code> from the <code>wordcloud</code> package	73
6.1	<code>as</code> and <code>wrap</code> declarations	75
6.2	Implicit use of <code>as</code> and <code>wrap</code>	75

6.3	Intrusive extension for <code>wrap</code>	77
6.4	Nonintrusive extension for <code>wrap</code>	77
6.5	Partial specialization for <code>wrap</code>	78
6.6	Intrusive extension for <code>as</code>	78
6.7	Nonintrusive extension for <code>as</code>	79
6.8	Partial specialization via <code>Exporter</code>	79
6.9	Partial specialization of <code>as</code> via <code>Exporter</code>	80
6.10	RcppBDT definitions of <code>as</code> and <code>wrap</code>	81
6.11	RcppBDT use of <code>as</code> and <code>wrap</code>	81
6.12	RcppBDT example for <code>getFirstDayOfWeekAfter</code>	82
7.1	A simple <code>norm</code> function in <code>C++</code>	84
7.2	Calling the <code>norm</code> function	84
7.3	A simple class <code>Uniform</code>	84
7.4	Exposing two member functions for <code>Uniform</code> class	85
7.5	Using the <code>Uniform</code> class from <code>R</code>	86
7.6	Exposing the <code>norm</code> function via modules	86
7.7	Using <code>norm</code> function exposed via modules	87
7.8	A module example with six functions	87
7.9	Modules example interface	87
7.10	Modules example use from <code>R</code>	88
7.11	Modules example with function documentation	88
7.12	Output for modules example with function documentation	89
7.13	Modules example with documentation and formal arguments	89
7.14	Output for modules example with documentation and formal arguments	89
7.15	Modules example with documentation and formal arguments without defaults	89
7.16	Usage of modules example with documentation and formal arguments	90
7.17	Modules example with ellipsis argument	90
7.18	Output of modules example with ellipsis argument	90
7.19	Exposing <code>Uniform</code> class using modules	91
7.20	Using <code>Uniform</code> class via modules	91
7.21	Constructor with a description	92
7.22	Constructor with a validator function pointer	92
7.23	Exposing fields and properties for modules	92
7.24	Field with documentation	93
7.25	Readonly-field with documentation	93
7.26	Property with getter and setter, or getter-only	93
7.27	Example of using a getter	94
7.28	Example of using a setter	94
7.29	Example code for properties	94
7.30	Example using properties	95
7.31	Example documenting a method	95
7.32	Const and non-const member functions	96

7.33	Example of S4 dispatch	96
7.34	Complete example of exposing <code>std::vector<double></code>	97
7.35	R use of <code>std::vector<double></code> modules example	98
7.36	R <code>NAMESPACE</code> import of Rcpp for modules	98
7.37	R <code>.onLoad()</code> code for module	98
7.38	Package skeleton support for modules	99
7.39	Use of <code>prompt</code> for documentation skeleton	100
7.40	NumPy load and save functions defined in RcppCNPY	100
7.41	Example of module declaration in RcppCNPY	102
8.1	A simple C++ function operating on vectors	103
8.2	A simple R function operating on vectors	104
8.3	A simple C++ function using sugar operating on vectors	104
8.4	Binary arithmetic operators for sugar	105
8.5	Binary logical operators for sugar	106
8.6	Unary operators for sugar	106
8.7	Functions returning a single boolean result	107
8.8	Using functions returning a single boolean result	107
8.9	Example using <code>is_na</code> sugar function	108
8.10	Example using <code>seq_along</code> sugar function	108
8.11	Example using <code>seq_len</code> sugar function	108
8.12	Example using <code>pmin</code> and <code>pmax</code> sugar function	109
8.13	Example using <code>ifelse</code> sugar function	109
8.14	Example using <code>sapply</code> sugar function	109
8.15	Example using <code>std::unary_function</code> functor with <code>sapply</code>	110
8.16	Example using <code>std::unary_function</code> functor with <code>mapply</code>	110
8.17	Example using <code>sign</code> sugar function	111
8.18	Example using <code>sign</code> sugar function	111
8.19	Example using <code>setdiff</code> sugar function	111
8.20	Example using <code>union</code> sugar function	111
8.21	Example using <code>intersect</code> sugar function	112
8.22	Example using <code>clamp</code> sugar function	112
8.23	Example using <code>unique</code> sugar function	112
8.24	Example using <code>table</code> sugar function	113
8.25	Example using <code>duplicated</code> sugar function	113
8.26	Examples using mathematical sugar functions	113
8.27	Examples of d/p/q/r statistical sugar functions sugar	114
8.28	Examples of using sugar RNG functions with <code>RNGScope</code>	115
8.29	The Curiously Recurring Template Pattern (CRTP)	117
8.30	The <code>VectorBase</code> class for Rcpp sugar	117
8.31	The <code>sapply</code> Rcpp sugar implementation	119
8.32	<code>Rcpp::traits::result_of</code> template	120
8.33	<code>result_of</code> trait implementation	120
8.34	<code>Rcpp::traits::r_sexptype_traits</code> template	120
8.35	<code>r_vector_element_converter</code> class	121
8.36	<code>storage_type</code> trait	121

8.37	Input expression base type	121
8.38	Output expression base type	121
8.39	Constructor for <code>Sapply</code> class template	122
8.40	Implementation of <code>Sapply</code>	122
8.41	Simulating π in <code>R</code>	123
8.42	Simulating π in <code>C++</code>	123
8.43	Simulating π in <code>R</code>	123
9.1	First <code>RInside</code> example: Hello, World!	128
9.2	Makefile for <code>RInside</code> examples	129
9.3	Using Makefile for <code>RInside</code> to build example	130
9.4	Second <code>RInside</code> example: data transfer	131
9.5	Third <code>RInside</code> example: data transfer	132
9.6	Fourth <code>RInside</code> example: plotting from <code>C++</code> via <code>R</code>	133
9.7	Fifth <code>RInside</code> example: parallel computing with <code>MPI</code>	134
10.1	A simple <code>Armadillo</code> example	139
10.2	<code>FastLm</code> function using <code>RcppArmadillo</code>	141
10.3	Basic <code>fLm()</code> function without formula interface	142
10.4	Basic <code>fastLmPure()</code> <code>R</code> function without formula interface ...	143
10.5	<code>FastLm</code> comparison	143
10.6	An example of a rank-deficient design matrix	144
10.7	Basic Kalman filter in <code>Matlab</code>	146
10.8	Basic Kalman filter in <code>R</code>	147
10.9	Basic Kalman filter in <code>R</code>	148
10.10	Basic Kalman filter class in <code>C++</code> using <code>Armadillo</code>	150
10.11	Basic Kalman filter function in <code>C++</code>	151
10.12	Basic Kalman filter timing comparison	151
10.13	Standard defines for <code>RcppArmadillo</code>	152
10.14	Standard defines for <code>RcppArmadillo</code>	153
11.1	<code>FastLm</code> function using <code>RcppGSL</code>	157
11.2	Definition of <code>gsl_vector</code> and <code>gsl_vector_int</code>	158
11.3	Example use of <code>gsl_vector</code>	159
11.4	Example use of <code>RcppGSL::vector<T></code>	159
11.5	Example <code>RcppGSL::vector<T></code> function	160
11.6	Example call of <code>RcppGSL::vector<T></code> function	160
11.7	Second example <code>RcppGSL::vector<T></code> function	160
11.8	Example call of second <code>RcppGSL::vector<T></code> function	161
11.9	Example of a vector view class	162
11.10	Example use <code>RcppGSL</code> matrix class	163
11.11	Implicit conversion for <code>RcppGSL</code> matrix class	163
11.12	Indexing for <code>RcppGSL</code> matrix class	164
11.13	Matrix norm in <code>R</code>	165
11.14	<code>Autoconf</code> script for <code>RcppGSL</code> use	165
11.15	Shell script configuration script for <code>RcppGSL</code> use	166
11.16	Windows shell script configuration script for <code>RcppGSL</code> use	166
11.17	Vector norm function for <code>RcppGSL</code>	166

11.18	Makevars.in for RcppGSL example	167
11.19	R function for RcppGSL example	168
11.20	Using RcppGSL with inline	168
11.21	Using package.skeleton with inline result	169
11.22	B-spline fit example from the GSL	169
11.23	Beginning of C++ file with B-spline fit for R	172
11.24	Data generation for GSL B-spline fit for R	172
11.25	Data fit for GSL B-spline with R	173
11.26	R side of GSL B-spline example	175
12.1	A simple Eigen example using fixed-size vectors and matrices	178
12.2	Eigen fixed-size vector and matrix representation	178
12.3	Eigen dynamic-size vector and matrix representation	179
12.4	A simple Eigen example using dynamic-size vectors and matrices	179
12.5	Comparing performance of simple operations between dynamic and fixed size vectors	179
12.6	Timing results simple operations between dynamic and fixed size vectors	180
12.7	Basic Kalman filter class in C++ using Eigen	182
12.8	Using a basic Eigen solver from R	184
12.9	Computing eigenvalues using Eigen	184
12.10	Computing least-squares using Eigen	185
12.11	Rank-revealing decompositions using Eigen	186
12.12	Core of definition of lm class in Eigen	187
12.13	Derived classes of lm providing specializations	188
12.14	Implementation of two subclass constructors for lm model fit	189
12.15	Selection of subclasses for lm model fit	189
12.16	Actual fastLm function in RcppEigen package	190
A.1	Simple C++ example: Hello, World!	195
A.2	Compiling and linking simple C++ example: Hello, World!	196
A.3	Compiling and linking simple C++ example in one step: Hello, World!	196
A.4	Simple C++ example using Rmath	196
A.5	Compiling and linking simple C++ example using Rmath	197
A.6	Simple R example of dynamic types	197
A.7	Simple R example of dynamic types	198
A.8	Simple C++ function example	199
A.9	Simple C++ function call example	200
A.10	Simple C++ data structure using struct	200
A.11	Simple C++ data structure using class	201
A.12	Simple C++ example using iterators on vector	202
A.13	Simple C++ example using const iterators on list	203
A.14	Simple C++ example using const iterators on deque	203
A.15	Simple C++ example using accumulate algorithm	203
A.16	Simple C++ template example	204
A.17	Another C++ template	204