Thesis for Degree of Master

Supervisor : Prof. Hyoung-Moon Kim

Making an R Package

Submitted by

Kexuan Yang

August, 2019

Department of Applied Statistics

Graduate School of Konkuk University

Making an R Package

A Master's Thesis
submitted to the Department of Applied Statistics
and the Graduate School of Konkuk University
in partial fulfillment of the
requirements for the degree of
Master of Arts

Submitted by

Kexuan Yang

May, 2019

This certifies that the Thesis of Kexuan Yang is approved.

Approved by Examination Committee

Chairman	______________________________
Member	______________________________
Member	______________________________

June, 2019

Graduate School of Konkuk University

TABLE OF CONTENTS

List of Figures

ABSTRACT

Making an R Package

Kexuan, Yang
Department of Applied Statistics
Graduate School of Konkuk University

This thesis gives a detailed introduction to making an R package for beginners of R. Starting with a few simple functions, I introduced the process of making an R package from beginning to end, it includes how to start making an R package, how to write metadata, and how to create your package and share it with others. In addition, based on EM algorithm, we used some new estimation methods for the Heckman sample selection model developed by Zhao et al. (2019). By combining these methods with how to make an R package, I made a package in order to use these algorithms and share them with other R users.

keyword : R package, sample selection, Heckman model, EM algorithm.

Chapter 1. Introduction

The R package is the foundation for the powerful scalability and flexibility of the R language and packs function-related functions for ease of use and distribution. It is a way to bundle R code, data, and documentation together, and it is easy to share with others.

Basically, the steps and descriptions about making an R package are introduced in R core team (1999), but for people, who are not familiar with using LaTeX, it can be an inconvenient experience. Also, Leisch (2008) introduced more simple ways for making an R package, but users still have to go through cumbersome experiences with using of LaTeX type of documentations. Recently, the book of making an R package easily is introduced in Wickham (2015). By this, making an R package is easier than you think, and the book describes how to make an R package in detail and explains the role of each file in the package by using some R functions and packages.

In chapter 2, the basic structure of a R package, and making a package with simple functions will be introduced. In chapter 3, for the Heckman selection model, we used some new estimation methods based on EM algorithm developed by Zhao et al. (2019). To share these methods with others and use these algorithms repeatedly, I made a package by combining the methods with how to make an R package. This package will be submitted to CRAN soon.

Chapter 2. A Simple Example for making an R Package

1. How to make an R package

The practical advantage to using a package is that it's easy and helpful to reload your code, because it's easy to share with R users and you can use it repeatedly. In this chapter, I will explain what is package and how to install it in section 1, and I will make a detailed easy toy example of making an R package in section 2.

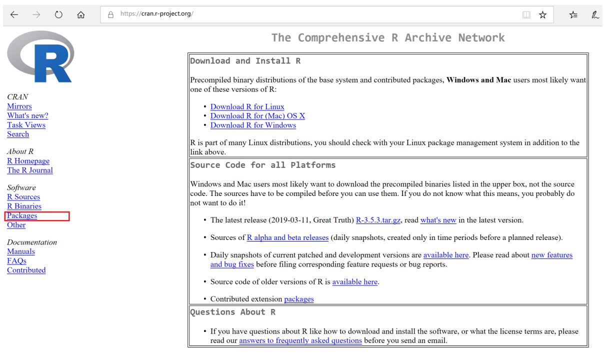
1.1. Preparations

Before making an R package, there are some preparations you need to do. Firstly, make sure you have a recent version of R and the latest version of RStudio. URLs are given as below:

1. download R:
   https://cran.r-project.org/
2. download RStudio:
   https://www.rstudio.com/products/rstudio/download/

My version of R was 3.5.1 and that of RStudio was 1.1.463 when I made this example. Next, it is necessary to know what's the package in R.

A package is a mechanism for extending the basic functionality of R, which integrates many functions. For instance, If you want to use a function that does not exist in the current R, you can find the package of the corresponding function to use directly. But where can you find packages? And how to use these packages? One of the methods is to download on CRAN($https : //cran.r - project.org/$). As shown in Figure 2-1, click on Packages in the menu bar on the left.


https://cran.r-project.org/

The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

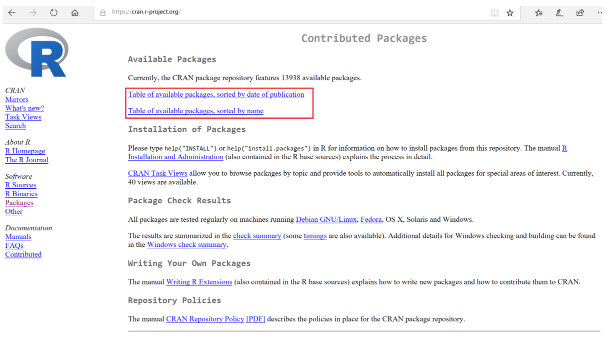
- The latest release (2019-03-11, Great Truth) R-3.5.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

<Figure 2-1> Download on CRAN

You can see that the first link is a list of available packages by release date; the second link is a list of packages by name as shown in Figure 2-2.


https://cran.r-project.org/

Contributed Packages

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Available Packages

Currently, the CRAN package repository features 13938 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 40 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, OS X, Solaris and Windows.

The results are summarized in the check summary (some timings are also available). Additional details for Windows checking and building can be found in the Windows check summary.

Writing Your Own Packages

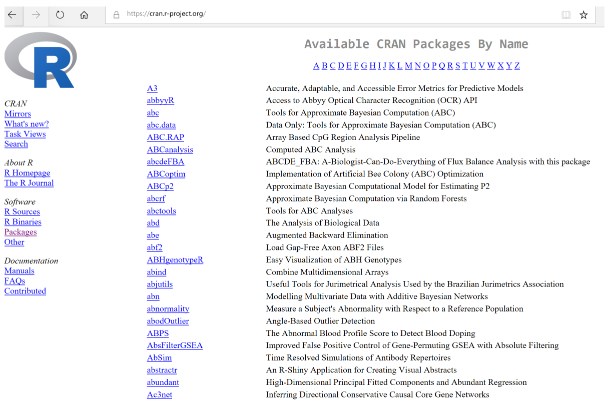
The manual Writing R Extensions (also contained in the R base sources) explains how to write new packages and how to contribute them to CRAN.

Repository Policies

The manual CRAN Repository Policy [PDF] describes the policies in place for the CRAN package repository.

<Figure 2-2> List of packages by name

Then click the second link to see: the left side of the list is the name of the package, and the right side is a brief description of the package, shown as Figure 2-3.



Available CRAN Packages By Name

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

A3	Accurate, Adaptable, and Accessible Error Metrics for Predictive Models
abbyyR	Access to Abbyy Optical Character Recognition (OCR) API
abc	Tools for Approximate Bayesian Computation (ABC)
abc.data	Data Only: Tools for Approximate Bayesian Computation (ABC)
ABC.RAP	Array Based CpG Region Analysis Pipeline
ABCanalysis	Computed ABC Analysis
abcdeFBA	ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package
ABCoptim	Implementation of Artificial Bee Colony (ABC) Optimization
ABCp2	Approximate Bayesian Computational Model for Estimating P2
abcrf	Approximate Bayesian Computation via Random Forests
abctools	Tools for ABC Analyses
abd	The Analysis of Biological Data
abe	Augmented Backward Elimination
abf2	Load Gap-Free Axon ABF2 Files
ABHgenotypeR	Easy Visualization of ABH Genotypes
abind	Combine Multidimensional Arrays
abjutils	Useful Tools for Jurimetrical Analysis Used by the Brazilian Jurimetrics Association
abn	Modelling Multivariate Data with Additive Bayesian Networks
abnormality	Measure a Subject's Abnormality with Respect to a Reference Population
abodOutlier	Angle-Based Outlier Detection
ABPS	The Abnormal Blood Profile Score to Detect Blood Doping
AbsFilterGSEA	Improved False Positive Control of Gene-Permuting GSEA with Absolute Filtering
AbSim	Time Resolved Simulations of Antibody Repertoires
abstractr	An R-Shiny Application for Creating Visual Abstracts
abundant	High-Dimensional Principal Fitted Components and Abundant Regression
Ac3net	Inferring Directional Conservative Causal Core Gene Networks

<Figure 2-3> Brief descriptions of packages

When using these packages, we generally do not download them, another easier way is to use the code($install.packages("packagename")$) directly in RStudio. and it will be described in the next section.

1.2. Installing required packages

After knowing the theory of R package, you need to install two necessary packages in advance: *devtools* and *roxygen*2, which are required for making R packages.

1. *devtools*

   Before making an R package, you need to spend time thinking about what you want your package to do, not the details of the package structure. According to Wickham (2015), the purpose of *devtools* is to achieve this philosophy. *devtools* makes packaging development as easy as possible, and when you focus on issues you are interested in, it can help us avoid many potential problems, especially in the development of packages. *devtools* works very closely with RStudio, it works with RStudio's existing code structure conventions and adds effective tools to support the loop of package development, so this development environment is very great for many R users: as long as you write a large amount of code, it will become your default layout.

2. *roxygen*2

One of the core requirements of the R package is that all exported functions, objects, and data sets are fully documented. RStudio includes several tools to help you create documentation, and *roxygen*2 is one of them. The *roxygen*2 package provides a source code documentation system that automatically generates *Rd* files. Although you can write *Rd* files directly in RStudio, it is much easier to use *roxygen*2 for package documentation and it is highly recommended for new package documentation. This will be shown in the toy example in the next section.

Now, let's start installing these two packages by using *install.packages("packagename")* in RStudio as an example.

Open RStudio, enter the code in the console: *install.packages*(*"devtools"*) and *install.packages*(*"roxygen*2*"*), these two packages will be downloaded automatically, the screen in the lower left corner shows the process of downloading and installing, and the red mark in the upper right corner of this window indicates that RStudion is currently busy and will be prompted to complete the download.

```
##install packages
install.packages("devtools")
install.packages("roxygen2")
```

Just downloading them is not directly usable, you need to load these packages before using them. The way to load the downloaded packages is to run *library*(*packagename*), in this case, run as follows:

```
###library
library(devtools)
library(roxygen2)
```

1.3. Creating a package

There are many ways of creating the package but I will introduce the one on the Wickham (2015), by using the RStudio, and there are five simple steps:

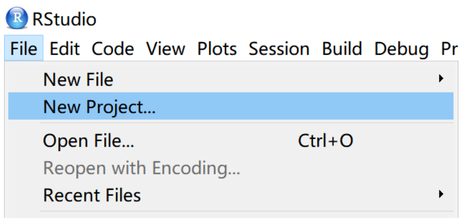
1. At first, choose *New Project* in *File* as shown in Figure 2-4.

2. Secondly, click *New Directory* as shown in Figure 2-5.

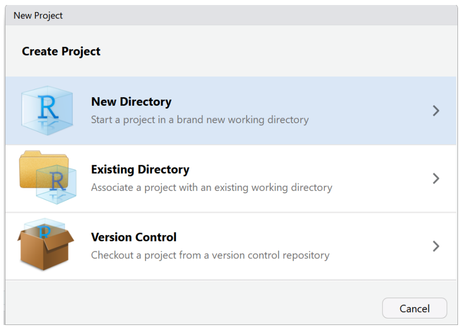
3. Thirdly, select R package, which is the third option as shown in Figure 2-6.

4. Fourthly, naming your package and changing the directory to wherever you want to locate the package.

5. Finally, if the session is restarted without the error, it is done. After all the five steps, the first file which is named as *hello.R* will be made automatically, as shown in Figure 2-8.



<Figure 2-4> Choose *New Project* in *File*

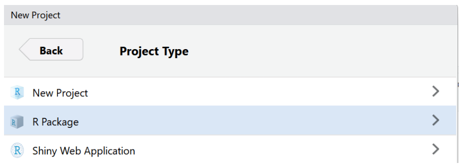


Start a project in a brand new working directory

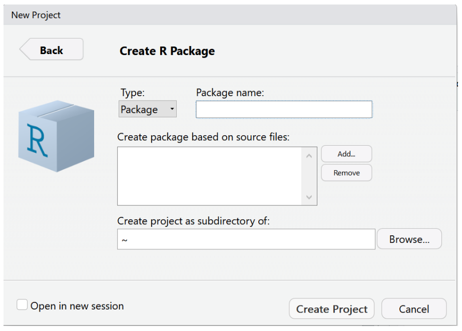
Associate a project with an existing working directory

Checkout a project from a version control repository

<Figure 2-5> Click *New Directory*

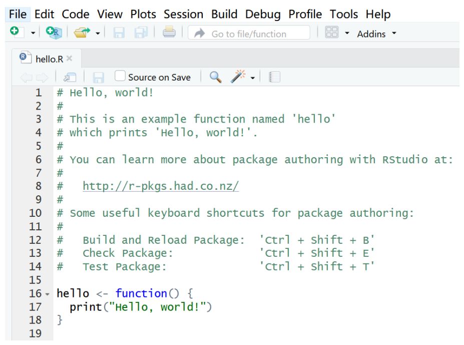


<Figure 2-6> Select "R package"

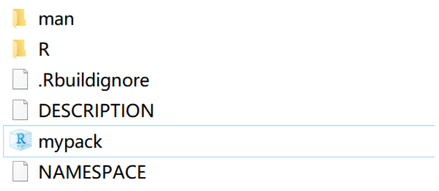


<Figure 2-7> Name the package

The new package you created will be stored in the document you specified and a series of default metadata files will be generated automatically. As shown in Figure 2-9: the job of the DESCRIPTION file is to store important metadata about your package, the code of *hello.R* is in *R/* which mentioned before and another code called *hello.Rd* is in a file called *man/*. At this point, the preparation is ready and a simple structure of the package was made.


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
hello.R
Source on Save

```
# Hello, world!
#
# This is an example function named 'hello'
# which prints 'Hello, world!'.
#
# You can learn more about package authoring with RStudio at:
#
#   http://r-pkgs.had.co.nz/
#
# Some useful keyboard shortcuts for package authoring:
#
#   Build and Reload Package:  'Ctrl + Shift + B'
#   Check Package:             'Ctrl + Shift + E'
#   Test Package:              'Ctrl + Shift + T'

hello <- function() {
  print("Hello, world!")
}
```



man
R
.Rbuildignore
DESCRIPTION
mypack
NAMESPACE

<Figure 2-8> Hello function R script <Figure 2-9> Package "mypack"

After the process of making a simple package is done, it is ready to locate your own functions, datasets, and other functions made out of other languages. Before locating these, make sure to remove some files located in the package that you made; *hello.R* and *hell.Rd* in */R* and */man* directory.

1.4. Description, Documentation, Namespace and Vignette (Metadata)

Next, the following is a brief introduction to the basic structure of the R package. The basic structure of the R package is as follows:

```
package (package name)
|
|--DESCRIPTION (Description file, including package name, version number,
title, description, dependencies, etc.)
|--NAMESPACE (Package namespace file)
|--R (Folder, function source code)
|---function1.R (R script file)
|---function2.R (R script file)
|--...
|--man (Folder, directory for storing functions' description files,
        help documents)
```

```
|---function1.Rd (R document file)
|---Package.Rd (R document file)
|--...
|--...
```

As it seen, there are some ingredients shown as above, and I will explain them with focuses on DESCRIPTION, Documentations, NAMESPACE, and Vignette.

DESCRIPTION

DESCRIPTION is a package description file, which is a plain text file that records information about the R package. It has no extension, so, it can be edited by any script editor. DESCRIPTION contains the following basic content, note that only a few fields are required, others are optional:

```
Package: package name
Type: Package
Title: What the package does
Version: 1.0
Date: when the package is built
Author: Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
Description: More about what it does (maybe more than one line)
License: Who can use your package?
```

Here, *package name* indicates the name of the package and *Type* : *Package* indicates the type of R-project, and the R package corresponds to "Package". The *Title* is when you take a title for your package, note that the title must includes a brief summary of the package. The *version* is used to record the version information of the package and the *Date* records the production time of it. *Maintainer* records the holder of the R package, and in the *Description*, you need to write more detailed introductions of your package, like what does your package want to do. Finally, the *License* indicates the license of your package.

Documentation

Documentation is another important part of a good package, if you need to introduce your package to others and let them know how to use it, write the detailed information here. After proper documentation, it is also like a dictionary, you can use $help()$ or ? to see a specific explanation of a function.

From Wickham (2015), *roxygen*2 is a package of R, modeled on the tools built by *doxygen*, to realize the idea that documents and source code are managed together with one file. This idea is quite good:

1. When writing code, it is convenient to write the document "in place" (function, parameter explanation, notes, dependencies and examples, etc.).
2. Update the document in time when the source code is modified

For editing the R package, the corresponding document is a file in the *.Rd* (R documentation) format. The following is a simple example of the framework:

```
#' product of two numbers.
#'
#' @param x A number.
#' @param y A number.
#' @return The product of \code{x} and \code{y}.
#' @examples
#' pro(1, 1)
#' pro(10, 1)
pro <- function(x, y) {
x * y
}
```

It will be shown more detail in section2, and I will use an example to explain what each line represents. and then a *man*/*pro.Rd* file will be generated by running *devtools* :: *document*(), that looks like:

```
% Generated by roxygen2: do not edit by hand
% Please edit documentation in R/samplepro.R
```

```
\name{pro}
\alias{pro}
\title{product of two numbers.}
\usage{
pro(x, y)
}
\arguments{
\item{x}{A number.}

\item{y}{A number.}
}
\value{
The product of \code{x} and \code{y}.
}
\description{
product of two numbers.
}
\examples{
pro(1, 1)
pro(10, 1)
}
```

Next, if you want to find the *help* file for the *pro* function, running ?*pro* or *help*("*pro*"), R will look for an *.Rd* file containing *alias* {"*pro*"}, then automatically parses the document and convert it to HTML to display the documentation. Here's the result in RStudio, as you can see, it looks like a LaTeX document, but it is slightly different.

```
pro {mypack}        R Documentation
product of two numbers.

Description

product of two numbers.

Usage

pro(x, y)
```

```
Arguments


x
A number.
y
A number.
Value

The product of x and y.

Examples

pro(1, 1)
pro(10, 1)
```

NAMESPACE

NAMESPACE is not that important if you are only developing package for yourself. But if you hope to submit your own package to CRAN you also need to understand it. The NAMESPACE of the package is a way for R package to manage objects within the package. It can help the author of the R package to control which function objects in the security are visible to the user, which objects are imported from other packages(import), and which objects are exported from the package(export).

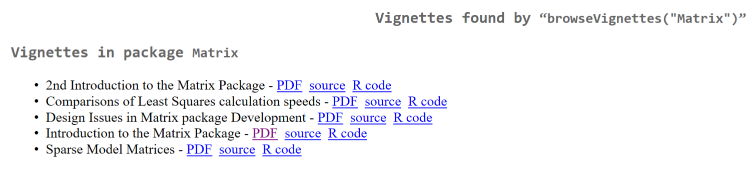
When you write an R package, sometimes there are many functions involved, but some of them may be abstract functions that are abstracted separately for the convenience of writing other functions. These sub-functions only need to be called by other functions in the R package, which is not helpful to external users, so they do not need to be visible to external users. The NAMESPACE file provides such a function. We can export functions that need to be visible to the user by creating a NAMESPACE file in the root of the package and by writing $export(functionname)$. Starting with R 2.14.0, all R packages must have a namespace, and if not, R will create it automatically.

Vignette

And a guide about your package is Vignette, a good Vignette can describe the problem that your package is designed to solve, and then show the reader how to solve it. To see the Vignette for a specific package, use the argument, `browseVignettes("packagename")`. For example:

```
browseVignettes("Matrix")
```

and you will see the vignette for *Matrix* package, it will provide three things: the original source file, a PDF file and R code as follows.



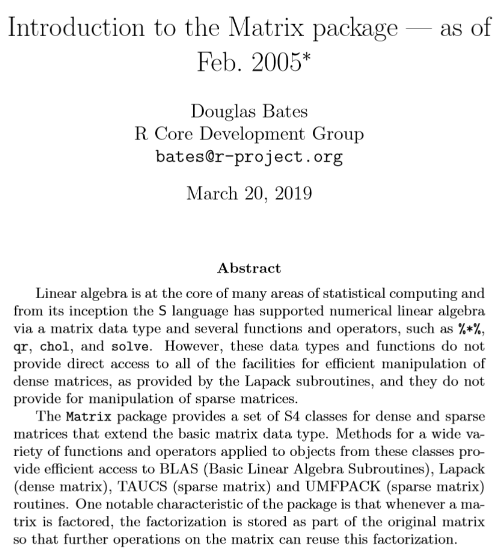
Vignettes found by "browseVignettes("Matrix")"

Vignettes in package Matrix

- 2nd Introduction to the Matrix Package - PDF source R code
- Comparisons of Least Squares calculation speeds - PDF source R code
- Design Issues in Matrix package Development - PDF source R code
- Introduction to the Matrix Package - PDF source R code
- Sparse Model Matrices - PDF source R code

<Figure 2-10> Vignette for *Matrix* package

For example, if you click the PDF file which in the fourth line, the Vignette was written by Bates (2019) will be shown as below.



Introduction to the Matrix package — as of Feb. 2005*

Douglas Bates
R Core Development Group
bates@r-project.org

March 20, 2019

Abstract

Linear algebra is at the core of many areas of statistical computing and from its inception the S language has supported numerical linear algebra via a matrix data type and several functions and operators, such as `%*%`, `qr`, `chol`, and `solve`. However, these data types and functions do not provide direct access to all of the facilities for efficient manipulation of dense matrices, as provided by the Lapack subroutines, and they do not provide for manipulation of sparse matrices.

The `Matrix` package provides a set of S4 classes for dense and sparse matrices that extend the basic matrix data type. Methods for a wide variety of functions and operators applied to objects from these classes provide efficient access to BLAS (Basic Linear Algebra Subroutines), Lapack (dense matrix), TAUCS (sparse matrix) and UMFPACK (sparse matrix) routines. One notable characteristic of the package is that whenever a matrix is factored, the factorization is stored as part of the original matrix so that further operations on the matrix can reuse this factorization.

<Figure 2-11> The PDF file for vignette

As you can see, a vignette normally looks like an academic paper or a chapter of a book, but it can be different by authors. It details the purpose, process, results, and practical application of the package you made. And the way to creat your vignette is as follows:

```
install.packages("rmarkdown")
usethis::use_vignette("mypack-vig")
```

Here, *rmarkdown* is a file format for creating dynamic documents in RStudio, and *Vignette* is written in the form of R markdown. For example, after running the above code, you will see:

1. a *vignettes* file will be created.
2. In the *vignettes* file, there will generate a $mypack - vig.Rmd$ file at the same time like this:

```
---
title: "Vignette Title"
author: "Vignette Author"
date: "‘r Sys.Date()‘"
output: rmarkdown::html_vignette
vignette: >
%\VignetteIndexEntry{Vignette Title}
%\VignetteEngine{knitr::rmarkdown}
%\VignetteEncoding{UTF-8}
---
```

3. There will be added the necessary dependencies in DESCRIPTION as follows:

```
Suggests:
knitr,
rmarkdown
VignetteBuilder: knitr
```

After editing your vignette, click knitr to knit the vignette and preview the output. That is, as in the example we mentioned above, after clicking knitr, you can generate a vignette like a paper.

1.5. Important functions in Devtools

Devtools functions can allow you to access tools from R instead of from command lines, and the purpose of it is to simplify the development of the package. All *devtools* functions accept paths as arguments, such as:
$load\_all("path/to/path/mypack")$.
The following are common development tasks for *Devtools*:

1. $devtools :: load\_all()$ simulates installing and reloading packages, loading R code in *.R*, compiling shared objects in *.src*, compiling in *.data*. $devtools :: load\_all$ will automatically create a DESCRIPTION if needed.

2. $devtools :: document()$ will update the documentation, file collation and NAMESPACE.

3. $devtools :: check()$ will update the document and then generate and check the package.

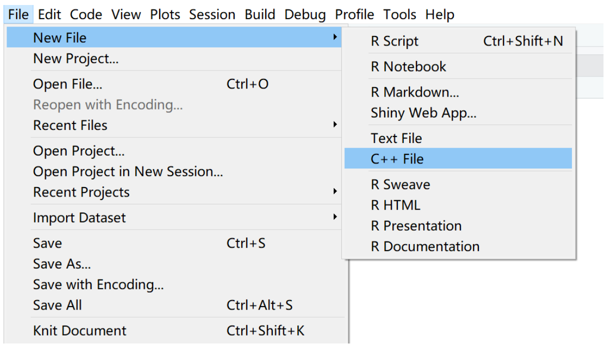
4. $devtools :: build()$ can generate a package file from the package source. You can use it to build a binary version of your package.

And they will be used in the next section.

1.6. C++ and other language codes

Although you have learned how to make an R package until the previous section and you have already felt that the R package is easy to develop on the function. However, the R function runs slower and often fails to meet the requirements for iterative calculations or Monte Carlo Markov Chains. In addition, since there are already many algorithm libraries in C or C++, people want to call the functions in these libraries directly. *Rcpp* was born to make it easier for R developers to use C++ functions. This section briefly describes how to write an R package that contains C++ functions.

Generate a C++ file with RStudio as shown in Figure 2-12.



<Figure 2-12> Generate a C++ file with RStudio

And the generated code template is as follows:

```
#include <Rcpp.h>
using namespace Rcpp;

// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar). Learn
// more about Rcpp at:
//
//   http://www.rcpp.org/
//   http://adv-r.had.co.nz/Rcpp.html
//   http://gallery.rcpp.org/
//

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
return x * 2;
}


// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.
//

/*** R
```

```
timesTwo(42)
*/
```

RStudio can quickly generate a standard code template as above. And the C++ function does not necessarily appear in your package, you can consider using it when there is a high demand for the speed of the function.

2. A Simple Example for Making an R Package

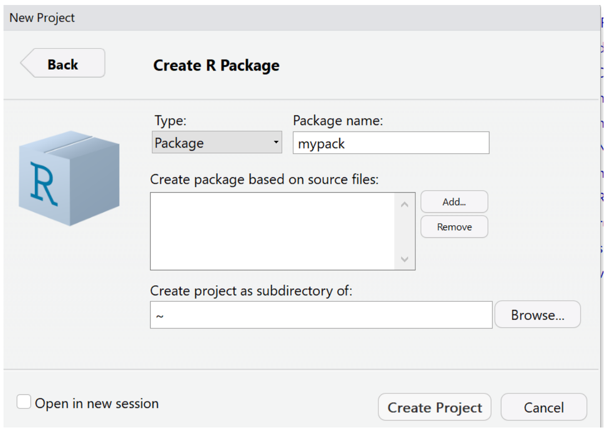
First of all, I will briefly introduce the making workflow of my package.

Workflow

1. Create a package.
2. Save the function that I want to share in the $\backslash R$.
3. Create a *data* file to include my own data.
4. Write DESCRIPTION.
5. Write Documentations about my functions in detail.
6. Build and check my package.

2.1. Making a null package

At first, we need a null package. As I introduced in previous section, we can make it on RStudio. I will make my package with the name called "mypack" as follows:



Type: Package
Package name: mypack
Create package based on source files:
Create project as subdirectory of: ~

<Figure 2-13> Generate a package: “mypack”

2.2. Functions

Let’s assume that I want to make a package, which contains functions calculating sample mean, sample correlation and sample covariance for given samples. Also assume that I want to save my own 1000 samples, which are randomly sampled from standard normal distribution. Firstly, we need functions, calculating sample mean, sample correlation and sample covariance for given samples. The following is the code of these functions.

```
sample.mean <- function(x){
sum(x)/length(x)
}
sample.variance <- function(x) {
sum((x-sum(x)/length(x))^2)/(length(x)-1)
}
sample.covariance <- function(x,y){
sum((x-sum(x)/length(x))*(y-sum(y)/length(y)))/(length(x)-1)
}
```

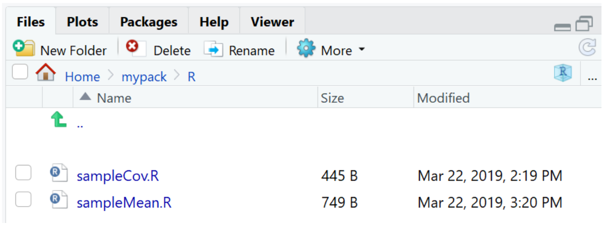
Here,

- `sample.mean` function calculates sample mean of given vector `x`.
- `sample.variance` function calculates sample variance of given vector `x`.

- `sample.covariance` function calculates the joint variability of two random variables `x` and `y`.

To put such functions to the package, I need to put and save them in R scripts in the directory mypack. Such scripts need to be created by names of such functions each by each. Each R script has to contain an unique function but in some cases it can be violated. I will talk about this case lately.

R scripts have to be located in the directory called *R* as shown in Figure 2-14.



```
Name           Size     Modified
..
sampleCov.R    445 B    Mar 22, 2019, 2:19 PM
sampleMean.R   749 B    Mar 22, 2019, 3:20 PM
```

<Figure 2-14> R scripts

But in this case, suppose I want to put the sample.mean and sample.variance functions together since they are the basic properties of sample, so in the documentation for the next section you will find two functions written together in an *.R* files.

2.3. Datasets

There are many built-in data sets in R, since R provides these data sets, you can call and use them for simulation experiments. Many data sets are contained in the R package called *datasets*, it is the basic package of R, which is in the search path of R, so these data sets can be directly applied. For example, you can call the built-in dataset *pressure* organized by Weast et al. (1973) as shown below:

```
> head(pressure)
temperature pressure
1           0    0.0002
2          20    0.0012
3          40    0.0060
4          60    0.0300
```

```
5          80    0.0900
6         100    0.2700
```

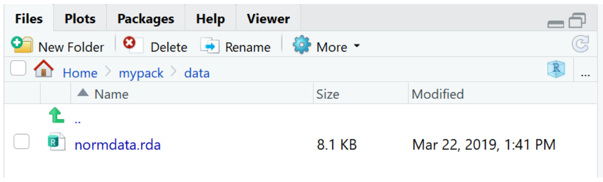
Data export in R package is also a matter of learning. If you want to release this package to anyone, then the data in the package is an extra bonus, but if you want to release it to R professionals, publishing data and documents together will make them more interested in your package and easier to operate.

Here I will introduce the way to include data into the package firstly, that is, put example datasets in *data/*. file. Also there are other ways to do this, and you can refer to the book(Wickham , 2015) for more details.

When you want to store binary data and share it with other uses, the best location for putting R package data is *data/*. file, under this folder, each *.RDA* file is automatically created by *save*() and has the same name as the file you created. In R, the best way to create a dataset is to use *usethis* :: *use_data*(). Suppose I want to put 1000 samples named normdata.

```
normdata <- rnorm(1000)
usethis::use_data(normdata)
```

After this step, a directory named *data* will be automatically generated, which means that the data file normdata created is automatically saved in this directory shown in Figure 2-15.



Name	Size	Modified
..		
normdata.rda	8.1 KB	Mar 22, 2019, 1:41 PM

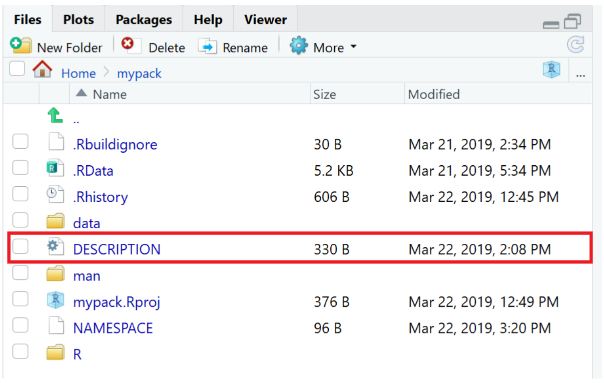
<Figure 2-15> Dataset: "normdata"

If you see *LazyData* : *true* in DESCRIPTION, which was generated by *usethis* :: *use_data*(), then the load data will be delayed, which means that the data does not occupy any memory.

2.4. Description, Documentation and Namespace (Metadata)

Description

According to the description section in Chapter 2, after the basic creation of the package, you can see the automatically generated description file in the directory, and this file describes and introduces your package, so you need to write enough details. For the examples, my description file should be written as follows. Note that the DESCRIPTION file can be edited directly in R Studio, by clicking such icon in the interface in Figure 2-16.



Name	Size	Modified
..		
.Rbuildignore	30 B	Mar 21, 2019, 2:34 PM
.RData	5.2 KB	Mar 21, 2019, 5:34 PM
.Rhistory	606 B	Mar 22, 2019, 12:45 PM
data		
DESCRIPTION	330 B	Mar 22, 2019, 2:08 PM
man		
mypack.Rproj	376 B	Mar 22, 2019, 12:49 PM
NAMESPACE	96 B	Mar 22, 2019, 3:20 PM
R		

<Figure 2-16> Description file

```
Package: mypack
Type: Package
Title: Test package for making R packages
Version: 0.1.0
Author: Yang Kexuan
Maintainer: Yang Kexuan <kexuanY@163.com>
Description: This package is for the simple description of
how to make R packages.
Depends: R (>= 2.10)
License: GPL-2
Encoding: UTF-8
LazyData: true
RoxygenNote: 6.1.1
LinkingTo:
Rcpp
```

```
Imports:
Rcpp
```

The contents in the above picture are explained as follows:

1. *title* and *description*

   The *title* and *description* fields describe the functionality of the package. The only difference is their length.

2. *version*

   An R package *version* is a sequence of at least two integers separated by either . or $-$, R uses the version number to determine if package dependencies are met. For instance, the package might import the package $devtools(>= 1.9.5)$, in which case version 1.9 or 1.9.2 will not work.

3. *Author* and *Maintainer*

   Using the *Author* to identify the author of your package, and whom to contact when a problem occurs. Here, author is Yang Kexuan, and my email address is kexuanY@163.com.

4. *depends*

   The *depends* in the eighth line is to require a specific version of R, for example, $Depends : R(>= 2.10)$. As with software packages, it is best to use it safely and require a version greater than or equal to the version you are currently using. Just run $devtools :: create()$ and it will do this for you.

5. *license*

   the *license* field can be a standard abbreviation for an open source license, such as $GPL - 2$ or BSD, or a pointer to a file containing more information, the file LICENSE. If you plan to release a package, the license is very important. If you don't, you can ignore this part.

6. *LazyData*

   LazyData provides easier access to the data in the package. Because of its importance, it is included in the minimal description created by devtools

In fact, there are still many options that are not used much. If you need a complete list, you can go to the "the description file" of the R extension manual. You can create your own fields, but the only restriction is that you can't use existing names. It's worth noting that if you plan to submit to CRAN, the name you use should be a valid English word.

Documentation

Since anonymous people you don't know will use your package, you need to describe the functions and datasets you made. There are multiple forms of documentations, and in this chapter I mainly want to introduce the object documentation. In the RStudio development tool, this process is done using `roxygen2` package, I will explain in detail how to write roxygen comments in this example. After you have written a documentation about your function, using the $devtools :: document()$ function mentioned in 1.5, the *.R* script file of your function will be automatically converted into an *.Rd* file and saved in the *man* directory. This documentation will be displayed when you use the function $help()$ in R. If it doesn't appear, make sure you are using devtools packages and that you have loaded your package with $devtool :: load\_all$.

`roxygen2` is used to make documenting your code more convenient and easier. By Wickham (2015), compared with writing *.Rd* files by hand, `roxygen2` has the following advantages:

1. The code and documentation are together, so when you modify the code, you will remember that the document needs to be updated.

2. `roxygen2` can dynamically inspect the record object while recording, so it can automatically add the written data during the writing process.

3. It describes the differences between the S3 and S4 methods, generics and classes, so you don't need to spend a lot of time learning the details.

`roxygen2` comments start with $\#'$, it can be used to distinguish the `roxygen2` comments from any normal R comments. Before the function, each row of `Roxygen` is called a block, and each block is treated as a tag, in the form of

`@tagName detailed description`. If you want to simply add a text description, you need to write two `@`.

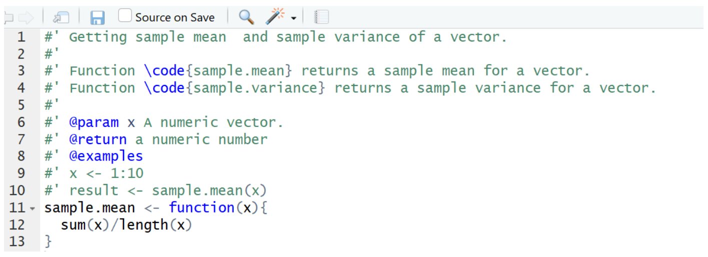
Before the first tag, every block contains some introductory text:

1. The first *sentence* is the title of your documentation, this is what you see at the top when you view the help file.

2. The second *paragraph* is the description: This part is mainly used to describe what your function does.

3. Starting from the third *paragraph*, it is a detailed introduction of your entire documentation. You should describe each argument carefully and explain how your function works.

My `Roxygen Comments` of the documentation is shown in Figure 2-17:



```
#' Getting sample mean  and sample variance of a vector.
#'
#' Function \code{sample.mean} returns a sample mean for a vector.
#' Function \code{sample.variance} returns a sample variance for a vector.
#'
#' @param x A numeric vector.
#' @return a numeric number
#' @examples
#' x <- 1:10
#' result <- sample.mean(x)
sample.mean <- function(x){
  sum(x)/length(x)
}
```

<Figure 2-17> `Roxygen Comments` of the documentation

As you can see:

1. "Getting sample mean and sample variance of a vector" is the title of the function.

2. In the second *paragraph*, there are descriptions of my function.

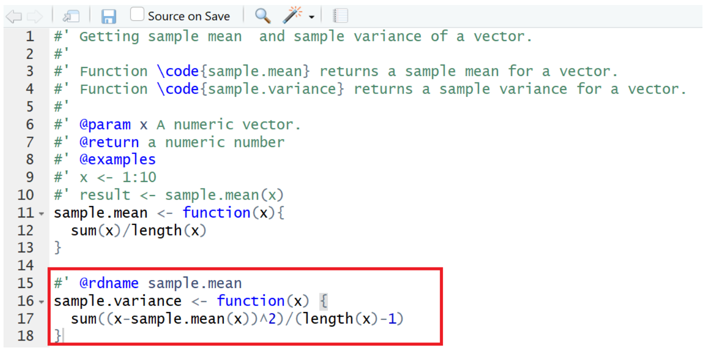
3. *@param* is used to interpret the parameters, which in my case is "a number".

4. *@return* is the value, it describes the output of the function.

5. *@example* showed how to use this function in practice. Since you are faced with a new R package, you will first look at these examples to know how to apply it, so it is very important in the DOCUMENTATION.

Here, I have only shown a simple example of documentation. A more detailed description will be shown in the specific example in Chapter 3.

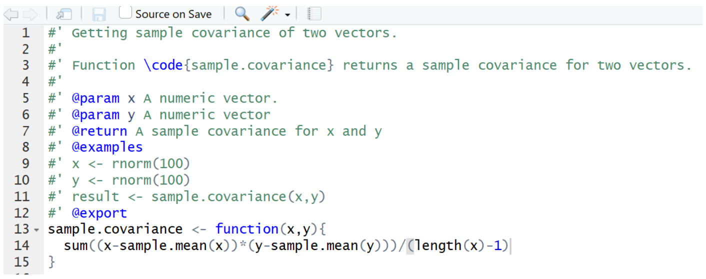
And if you want to combine multiple functions that are similar in usage in one package together, *@rdname* can help you solve this problem. You just need to mark multiple functions as a common *@rdname*, but if *@rdname* is used, then *@description* and *@details* need to be written in the first function, and *@title* will only appear in the first function, too. In my case, the code is as below:



```
#' Getting sample mean  and sample variance of a vector.
#'
#' Function \code{sample.mean} returns a sample mean for a vector.
#' Function \code{sample.variance} returns a sample variance for a vector.
#'
#' @param x A numeric vector.
#' @return a numeric number
#' @examples
#' x <- 1:10
#' result <- sample.mean(x)
sample.mean <- function(x){
  sum(x)/length(x)
}

#' @rdname sample.mean
sample.variance <- function(x) {
  sum((x-sample.mean(x))^2)/(length(x)-1)
}
```

<Figure 2-18> Add another function

And the documentation of sample.covariance function is also shown as follows:



```
#' Getting sample covariance of two vectors.
#'
#' Function \code{sample.covariance} returns a sample covariance for two vectors.
#'
#' @param x A numeric vector.
#' @param y A numeric vector
#' @return A sample covariance for x and y
#' @examples
#' x <- rnorm(100)
#' y <- rnorm(100)
#' result <- sample.covariance(x,y)
#' @export
sample.covariance <- function(x,y){
  sum((x-sample.mean(x))*(y-sample.mean(y)))/(length(x)-1)
}
```

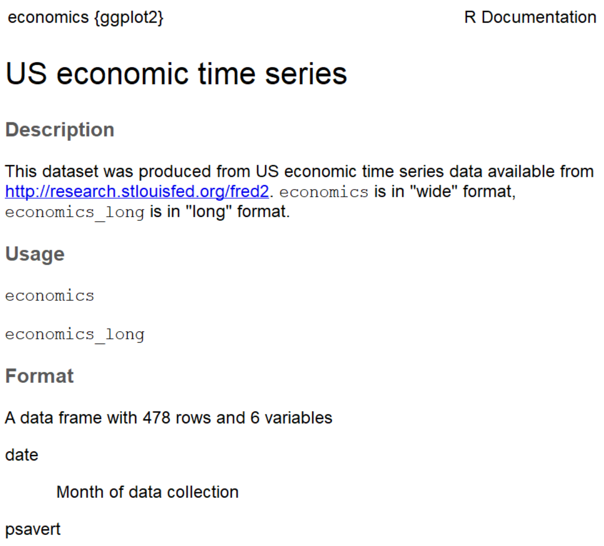
<Figure 2-19> Documentation of sample.covariance function

Similarly, you need to document the dataset, just like documenting a function.

You can name the dataset and document it in *R/.* instead of directly documenting the data, and all the *roxygen*2 code for datasets can be included in a single *R/.* file in the R directory of the package. See for examples, the *economics* dataset in Wickham (2016).

```
#' US economic time series.
#'
#' This dataset was produced from US economic time
series data available from
\url{http://research.stlouisfed.org/fred2}.
#'
#' \itemize{
#'   \item date.  Month of data collection
#'
#'   \item psavert, personal savings rate,
#' ...
#' }
#'
#' @docType data
#' @keywords datasets
#' @name economics
#' @usage data(economics)
#' @format A data frame with 478 rows and 6 variables
NULL
```

And the result of this in a help file that looks like as follows:


economics {ggplot2} R Documentation

US economic time series

Description

This dataset was produced from US economic time series data available from http://research.stlouisfed.org/fred2. `economics` is in "wide" format, `economics_long` is in "long" format.

Usage

```
economics

economics_long
```

Format

A data frame with 478 rows and 6 variables

date

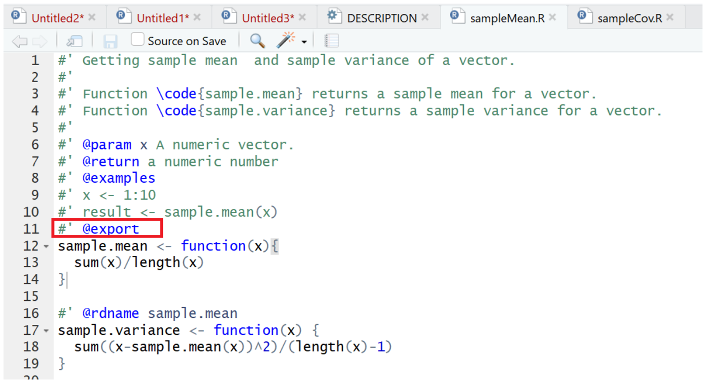
Month of data collection

psavert

<Figure 2-20> Help file of *economics*

Namespace

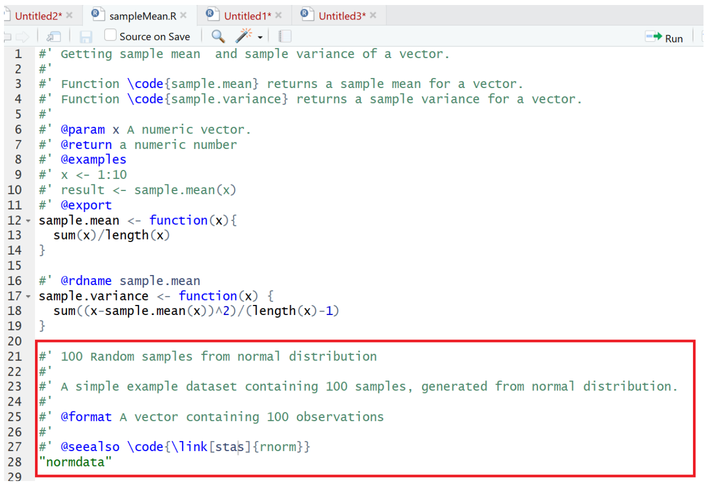
As the name implies, the namespace provides "spaces" for "names." They provide a context for finding the value of an object associated with a name. So in order to evade the collaption and confusion between function you have to add namespace. Here, add #′ @export to the end of your *roxygen* chunk:



```
#' Getting sample mean  and sample variance of a vector.
#'
#' Function \code{sample.mean} returns a sample mean for a vector.
#' Function \code{sample.variance} returns a sample variance for a vector.
#'
#' @param x A numeric vector.
#' @return a numeric number
#' @examples
#' x <- 1:10
#' result <- sample.mean(x)
#' @export
sample.mean <- function(x){
  sum(x)/length(x)
}

#' @rdname sample.mean
sample.variance <- function(x) {
  sum((x-sample.mean(x))^2)/(length(x)-1)
}
```

<Figure 2-21> Add #′ @export

And then document the name of the dataset as below.



```
#' Getting sample mean  and sample variance of a vector.
#'
#' Function \code{sample.mean} returns a sample mean for a vector.
#' Function \code{sample.variance} returns a sample variance for a vector.
#'
#' @param x A numeric vector.
#' @return a numeric number
#' @examples
#' x <- 1:10
#' result <- sample.mean(x)
#' @export
sample.mean <- function(x){
  sum(x)/length(x)
}

#' @rdname sample.mean
sample.variance <- function(x) {
  sum((x-sample.mean(x))^2)/(length(x)-1)
}

#' 100 Random samples from normal distribution
#'
#' A simple example dataset containing 100 samples, generated from normal distribution.
#'
#' @format A vector containing 100 observations
#'
#' @seealso \code{\link[stats]{rnorm}}
"normdata"
```

<Figure 2-22> Documentation of the dataset

There is an important additional tag for documenting datasets: *@format* outlines the dataset and for data frames, you should include a list of definitions that describe each variable.

2.5. Using Rcpp with RStudio

As mentioned before, although R language has great advantages in data statistics and drawing, the slow operation is its biggest shortcoming, especially for the case of large data volume. Using the underlying speed advantage of C++, using R language to call C++ functions can effectively compensate for this defect of R. In this section, we will continue to use examples to illustrate how to use *Rcpp* with RStudio.

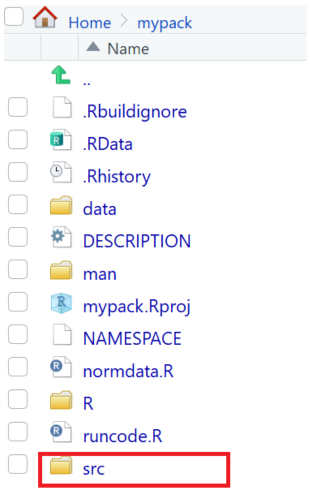
1. First of all, in order to create a file based on *Rcpp* associated with C++, run the code as follows:

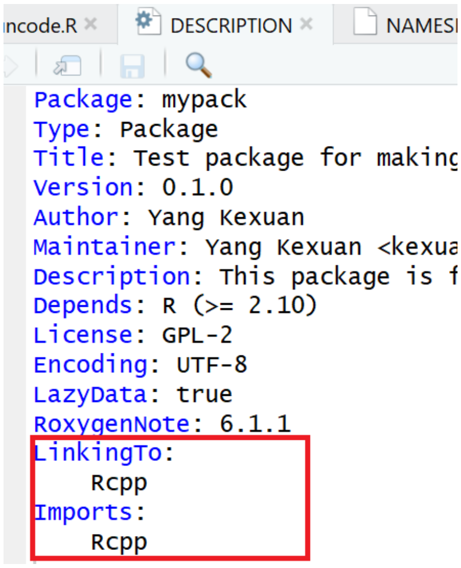
   ```
   devtools::use_rcpp()
   ```

   After running, you will see the following results:

   (a) A new *src* directory will be generated in your folder as shown in Figure 2-23.

(b) *Rcpp* was automatically added to the *Linking* and *Imports* files in the DESCRIPTION. It's shown in Figure 2-24.

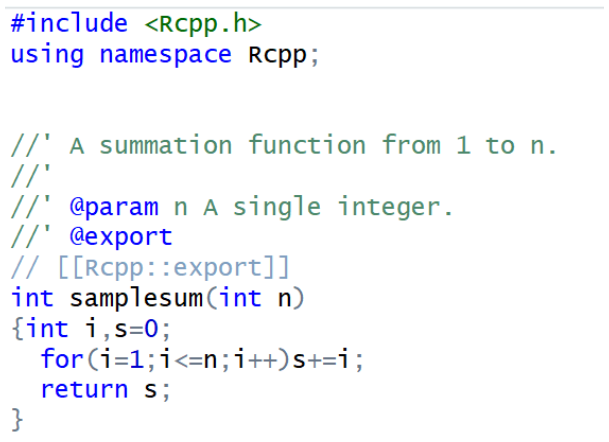




```
Package: mypack
Type: Package
Title: Test package for makinc
Version: 0.1.0
Author: Yang Kexuan
Maintainer: Yang Kexuan <kexua
Description: This package is f
Depends: R (>= 2.10)
License: GPL-2
Encoding: UTF-8
LazyData: true
RoxygenNote: 6.1.1
LinkingTo:
    Rcpp
Imports:
    Rcpp
```

<Figure 2-23> *Src* directory <Figure 2-24> *Linking* and *Imports*

2. Then, creating a C++ file as described in Section 1.6. There, I designed a sum function to find the sum from 1 to n. of course, you can add any function you want to create here. Figure 2-25 shows how I did it. And noting that it should be stored in the *src* file.



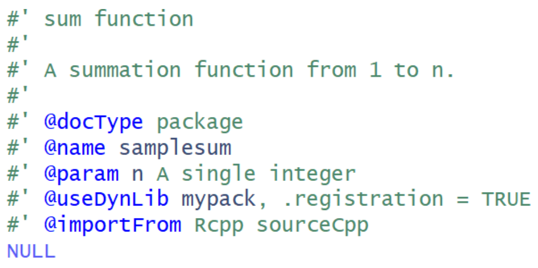
```
#include <Rcpp.h>
using namespace Rcpp;


//' A summation function from 1 to n.
//'
//' @param n A single integer.
//' @export
// [[Rcpp::export]]
int samplesum(int n)
{int i,s=0;
  for(i=1;i<=n;i++)s+=i;
  return s;
}
```

<Figure 2-25> C++ function

3. Next, like making a general package, we need to have a DOCUMENTATION to describe my C++ function. As mentioned before, this file should be saved in *R* folder. There is also something different from the general package, we need to add the following code:

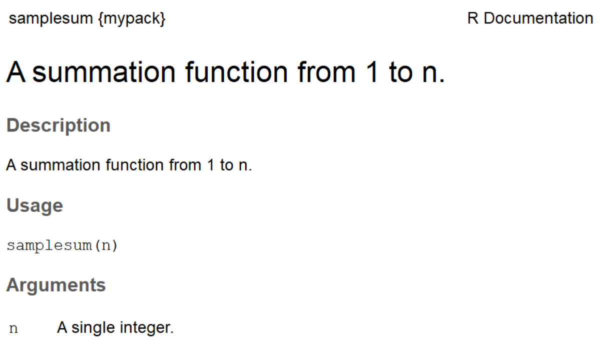
```
#' @useDynLib your package name, .registration = TRUE
#' @importFrom Rcpp sourceCpp
```

My DOCUMENTATION about C++ function is shown in Figure 2-26. And run ?*samplesum* you will see the *help* file as shown in Figure 2-27 like general R packages.



```
#' sum function
#'
#' A summation function from 1 to n.
#'
#' @docType package
#' @name samplesum
#' @param n A single integer
#' @useDynLib mypack, .registration = TRUE
#' @importFrom Rcpp sourceCpp
NULL
```

<Figure 2-26> Documentation of the C++ function



samplesum {mypack} R Documentation

A summation function from 1 to n.

Description

A summation function from 1 to n.

Usage

```
samplesum(n)
```

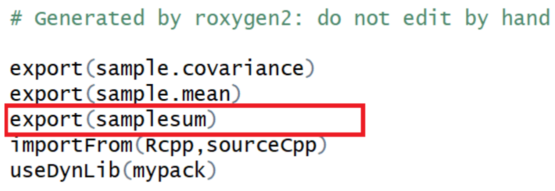
Arguments

n	A single integer.

<Figure 2-27> *Help* file of the C++ function

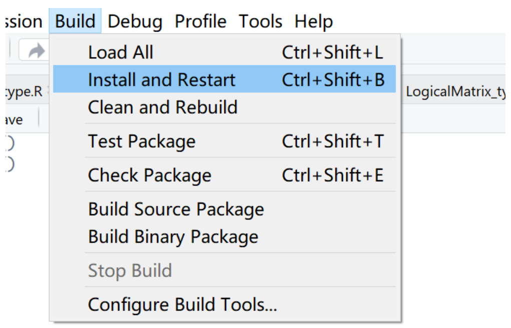
4. After creating *Rd* and *cpp* these two files, run the following code on the RStudio console to generate the necessary modifications to the NAMESPACE, and you will see the changes in NAMESPACE in Figure 2-28.

```
devtools::document()
devtools::load_all()
```



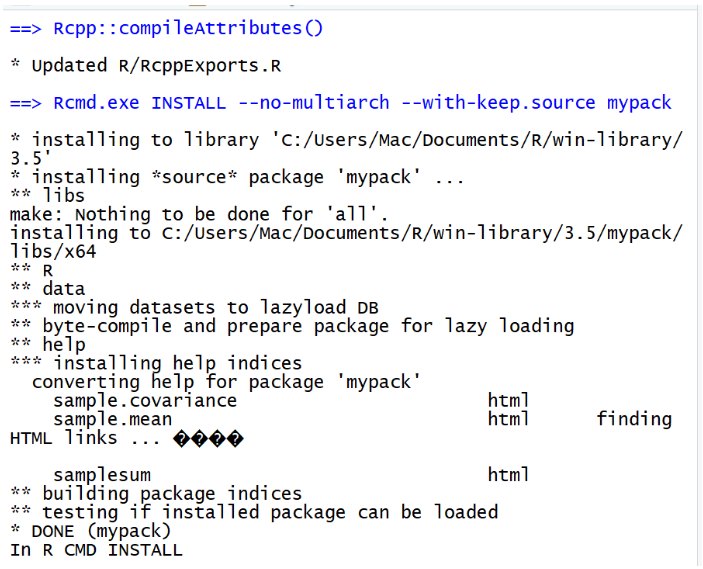
```
# Generated by roxygen2: do not edit by hand

export(sample.covariance)
export(sample.mean)
export(samplesum)
importFrom(Rcpp,sourceCpp)
useDynLib(mypack)
```

<Figure 2-28> Namespace of the C++ function



ssion Build Debug Profile Tools Help
Load All Ctrl+Shift+L
Install and Restart Ctrl+Shift+B
Clean and Rebuild
Test Package Ctrl+Shift+T
Check Package Ctrl+Shift+E
Build Source Package
Build Binary Package
Stop Build
Configure Build Tools...

<Figure 2-29> Install and restart



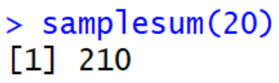
```
==> Rcpp::compileAttributes()

* Updated R/RcppExports.R

==> Rcmd.exe INSTALL --no-multiarch --with-keep.source mypack

* installing to library 'C:/Users/Mac/Documents/R/win-library/
3.5'
* installing *source* package 'mypack' ...
** libs
make: Nothing to be done for 'all'.
installing to C:/Users/Mac/Documents/R/win-library/3.5/mypack/
libs/x64
** R
** data
*** moving datasets to lazyload DB
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
  converting help for package 'mypack'
    sample.covariance                       html
    sample.mean                             html      finding
HTML links ... ����

    samplesum                               html
** building package indices
** testing if installed package can be loaded
* DONE (mypack)
In R CMD INSTALL
```

<Figure 2-30> Running process

and click *Install&Restart* in the same pane as shown in Figure 2-29, the running process will be like in Figure 2-30, if there is no warning, let's run the C++ function we just created to verify success. For example, run

```
samplesum(20)
```

The running results is shown in Figure 2-31



```
> samplesum(20)
[1] 210
```

<Figure 2-31> Running result

2.6. Checking and building

After all the documentations have been completed, use the devtools functions to update and load the package.

```
devtools::document()
devtools::load_all()
```

The following results will appear in the lower left corner of the RStudio interface.

```
> devtools::document()
Updating mypack documentation
Writing NAMESPACE
Loading mypack
Writing NAMESPACE
> devtools::load_all()
Loading mypack
```

Finally, if you want to submit your package to CRAN, you need to build the package and check it for common problems. *devtools* :: *build* is used to generate a package file and *devtools* :: *check* is used to automatically check the code in R. It's still helpful to find problems in your package and fix them even if you don't plan to submit it, because some problems are hard to find. The procedures for building and checking are as follows:

1. Use *devtools* :: *check* to check your package on the platform.
2. Use *devtools* :: *build* to prepare your source package and don't forget to confirm that you have added your e-mail address in the "Maintainer" field of the DESCRIPTION file as mentioned in DESCRIPTION section.

As you can see, the procedures of building and checking are simple and automatic. In my case, the procedure results are as follows.

The results of building are shown in Figure 2.28, it will generate a file named "$mypack\_0.1.0.tar.gz$".

```
> devtools::check()
Updating mypack documentation
Writing NAMESPACE
Loading mypack
Writing NAMESPACE
-- Building --------------------------------- mypack --
Setting env vars:
* CFLAGS    : -Wall -pedantic
* CXXFLAGS  : -Wall -pedantic
* CXX11FLAGS: -Wall -pedantic
-------------------------------------------------------
√  checking for file
'C:\Users\Mac\Documents\mypack/DESCRIPTION' (477ms)
-  preparing 'mypack': (471ms)
√  checking DESCRIPTION meta-information ...
-  cleaning src
-  checking for LF line-endings in source and make files and
shell scripts
-  checking for empty or unneeded directories
-  looking to see if a 'data/datalist' file should be added
-  building 'mypack_0.1.0.tar.gz'
-- Checking ----------------------------
 using log directory
'C:/Users/Mac/AppData/Local/Temp/RtmpiGKXs9/mypack.Rcheck' (351ms)
-  using R version 3.5.1 (2018-07-02)
...
** checking whether the package can be loaded ... OK
** checking whether the package can be loaded with stated
dependencies ... OK
** checking whether the package can be unloaded cleanly ... OK
** checking whether the namespace can be loaded with stated
dependencies ... OK
** checking whether the namespace can be unloaded cleanly ... OK
** checking loading without being on the library search path ... OK
√  checking dependencies in R code ...
√  checking S3 generic/method consistency (724ms)
√  checking replacement functions (418ms)
...
** running examples for arch 'i386' ... OK
** running examples for arch 'x64' ... OK
DONE
-- R CMD check results ----------------- mypack 0.1.0 ----
Duration: 49.8s

0 errors √ | 0 warnings √ | 0 notes √
```

```
> devtools::build()
√  checking for file 'C:\Users\Mac\Documents\mypack/DESCRIPTION' ...
-  preparing 'mypack': (779ms)
√  checking DESCRIPTION meta-information ...
-  cleaning src
-  checking for LF line-endings in source and make files and
   shell scripts
-  checking for empty or unneeded directories
-  looking to see if a 'data/datalist' file should be added
-  building 'mypack_0.1.0.tar.gz'

[1] "C:/Users/Mac/Documents/mypack_0.1.0.tar.gz"
```

and the results of checking are as above.

If there are no errors or warnings in the running results as above, you can post your package online or elsewhere. You can choose to compress your package folder and give it to others for local installation, or simpler to download directly from Github via RStudio.

Chapter 3. R Package for Sample Selection Model

1. Introduction

Based on Zhao et al. (2019), some new algorithms: `ECM` and `ECMnr` are used for estimating model parameters of Heckman model based on `EM` algorithm, they still retain the appealing properties about easy implementation and numerically stable of `EM` algorithm. It can be seen from the simulation results that these new algorithms achieves more robust and better estimation results by the `EM` algorithm. First of all, I want to briefly introduce Heckman selection model, and then introduce the two new algorithms in detail.

2. Heckman selection model

Assuming that Y_{i1} is the response variable and Y_{i2} is the selection variable, $Y_{i1} = \mathbf{x}_i^\top \boldsymbol{\beta} + \epsilon_i,\ i = 1, \dots, N$ is the standard regression form, and $Y_{i2} = \mathbf{w}_i^\top \gamma + \eta_i,\ i = 1, \dots, N$ is the selection equation. And $u_i = I(y_{i2} > 0)$ because observing only N_1 out of N observations y_{i1} for which $y_{i2} > 0$. There are unknown parameters $\boldsymbol{\beta} \in \mathbb{R}^p$ and $\gamma \in \mathbb{R}^q$, known characteristics $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{w}_i \in \mathbb{R}^q$. By assuming bivariate normality in the error terms ϵ_i and η_i (Heckman, 1974), thus

$$\begin{pmatrix} Y_{i1} \\ Y_{i2} \end{pmatrix} \sim \text{independently } N_2 \left[\begin{pmatrix} \mathbf{x}_i^T \boldsymbol{\beta} \\ \mathbf{w}_i^T \gamma \end{pmatrix}, \begin{pmatrix} \sigma^2 & \rho\sigma \\ \rho\sigma & 1 \end{pmatrix} \right]. \tag{1}$$

In matrix notation, $\mathbf{y} = \mathbf{X}^* \boldsymbol{\beta}^* + \boldsymbol{\epsilon}$ is equivalent to

$$\begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{X} & \mathbf{0} \\ \mathbf{0} & \mathbf{W} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ \gamma \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \end{pmatrix},$$

where

$$\boldsymbol{\epsilon} \sim N_{2n}(\mathbf{0}, \Sigma \otimes I_N), \Sigma = \begin{pmatrix} \sigma^2 & \rho\sigma \\ \rho\sigma & 1 \end{pmatrix}.$$

The likelihood function for the observed data is:

$$L(\boldsymbol{\theta}; \mathbf{y}, \mathbf{u}) = \prod_{i=1}^{N} \left[f(y_{i1}|u_i = 1) P(U_i = 1) \right]^{u_i} \left[P(U_i = 0) \right]^{1-u_i}, \boldsymbol{\theta} = \{\boldsymbol{\beta}, \gamma, \sigma, \rho\}, \tag{2}$$

where

$$f(y_{i1}|u_i = 1) = \frac{1}{\sigma}\phi\left(\frac{y_{i1} - \mathbf{x}_i^\top \boldsymbol{\beta}}{\sigma}\right) \frac{\Phi\left\{ \frac{\rho}{\sqrt{1-\rho^2}} \left(\frac{y_{i1} - \mathbf{x}_i^\top \boldsymbol{\beta}}{\sigma} \right) + \frac{\mathbf{w}_i^\top \gamma}{\sqrt{1-\rho^2}} \right\}}{\Phi(\mathbf{w}_i^\top \gamma)}, \tag{3}$$

where $\phi(\cdot)$ is the standard normal density function and $\Phi(\cdot)$ is the standard normal cumulative distribution function, and

$$P(U_i = u_i) = \left\{ \Phi(\mathbf{w}_i^\top \gamma) \right\}^{u_i} \left\{ \Phi(-\mathbf{w}_i^\top \gamma) \right\}^{1-u_i}, u_i = 0, 1.$$

The density of (3) is known as the extended skew-normal distribution (Arellano–Valle et al., 2006).

3. Estimation methods

3.1. Existing estimation methods

By maximizing the log-likelihood of (2), the maximum likelihood estimators can be obtained. It's efficient when the distributional assumption of bivariate normality is correct.

The 2-step estimation method was suggested by Heckman (1979) is more robust to the normality assumption and is based on the following expectation:

$$E(Y_{i1}|Y_{i2} > 0) = \mathbf{x}_i^\top \boldsymbol{\beta} + \rho\sigma\lambda(\mathbf{w}_i^\top \gamma), \tag{4}$$

where $\lambda = \phi(\cdot)/\Phi(\cdot)$ denotes the inverse Mills ratio.

The third method was proposed by Little and Rubin (2002, pp. 322-323), that is EM algorithm. In general, the selection model contains the exclusion case as well, and thus the EM algorithm cannot be used. Three new algorithms for general cases based on the EM algorithm are developed in the following sections, thus they still remain the advantages: numerical stability and easy implementation.

3.2. New ECM algorithm

The new `ECM` algorithm was developed based on `ECM` algorithm (Meng and Rubin , 1993) which is an extension of `EM` algorithm. Assuming that the complete data $\mathbf{y} = (\mathbf{y}_1^T, \mathbf{y}_2^T)^T$, where $\mathbf{y}_j = (y_{1j}, y_{2j}, ..., y_{Nj})^T, j = 1, 2.$, and the related parameters are $\boldsymbol{\theta} = (\boldsymbol{\beta}, \gamma, \sigma, \rho)$. Generally, assuming the missing data is $\mathbf{y}_{mis} = ((y_{N_1+1,1}, \cdots, y_{N,1}), \mathbf{y}_2^\top)^\top$, and the observed data is $\mathbf{y}_{obs} = (y_{11}, \cdots, y_{N_1,1})^\top$. By the invariance property of MLEs,

$$\sigma^2 = \exp(\psi^*) + (\rho^*)^2 \text{ and } \rho = \frac{\rho^*}{\sqrt{\exp(\psi^*) + (\rho^*)^2}}, \tag{5}$$

where $\boldsymbol{\theta}^* = \{\boldsymbol{\beta}, \gamma, \psi^*, \rho^*\}$. And the complete data log-likelihood is

$$\begin{aligned} l_c(\boldsymbol{\theta}^*|\mathbf{y}) = &-N\log(2\pi) - \frac{N}{2}\log(\psi) - \frac{1}{2\psi}\left\{\sum_{i=1}^N (y_{i1} - \mathbf{x}_i^\top \boldsymbol{\beta})^2\right\} \\ &- \frac{1}{2}\left(1 + \frac{(\rho^*)^2}{\psi}\right)\left\{\sum_{i=1}^N (y_{i2} - \mathbf{w}_i^\top \gamma)^2\right\} + \frac{\rho^*}{\psi}\left\{\sum_{i=1}^N (y_{i1} - \mathbf{x}_i^\top \boldsymbol{\beta})(y_{i2} - \mathbf{w}_i^\top \gamma)\right\}. \end{aligned}$$

The corresponding Q-function at the k-th iteration is obtained as

$$Q\left(\boldsymbol{\theta}^*|\hat{\boldsymbol{\theta}}^{*(k)}\right) = E\left[l_c(\boldsymbol{\theta}^*|\mathbf{y})|\hat{\boldsymbol{\theta}}^{*(k)}\right] = -N\log(2\pi) - \frac{N}{2}\log(\psi)$$

$$- \frac{1}{2\psi}\left\{\sum_{i=1}^{N_1}(y_{i1} - \mathbf{x}_i^\top\boldsymbol{\beta})^2 + \sum_{i=N_1+1}^{N}(\hat{v}_{1m}^{(k)} - 2\hat{\alpha}_{1m}^{(k)}\mathbf{x}_i^\top\boldsymbol{\beta} + \boldsymbol{\beta}^\top\mathbf{x}_i\mathbf{x}_i^\top\boldsymbol{\beta})\right\}$$

$$- \frac{1}{2}\left(1 + \frac{(\rho^*)^2}{\psi}\right)\left\{\sum_{i=1}^{N_1}(\hat{v}_{2o}^{(k)} - 2\hat{\alpha}_{2o}^{(k)}\mathbf{w}_i^\top\gamma + \gamma^\top\mathbf{w}_i\mathbf{w}_i^\top\gamma)\right.$$

$$\left. + \sum_{i=N_1+1}^{N}(\hat{v}_{2m}^{(k)} - 2\hat{\alpha}_{2m}^{(k)}\mathbf{w}_i^\top\gamma + \gamma^\top\mathbf{w}_i\mathbf{w}_i^\top\gamma)\right\}$$

$$+ \frac{\rho^*}{\psi}\left\{\sum_{i=1}^{N_1}(y_{i1}\hat{\alpha}_{2o}^{(k)} - y_{i1}\mathbf{w}_i^\top\gamma - \hat{\alpha}_{2o}^{(k)}\mathbf{x}_i^\top\boldsymbol{\beta} + \boldsymbol{\beta}^\top\mathbf{x}_i\mathbf{w}_i^\top\gamma)\right.$$

$$\left. \sum_{i=N_1+1}^{N}(\hat{\alpha}_{12m}^{(k)} - \hat{\alpha}_{1m}^{(k)}\mathbf{w}_i^\top\gamma - \hat{\alpha}_{2m}^{(k)}\mathbf{x}_i^\top\boldsymbol{\beta} + \boldsymbol{\beta}^\top\mathbf{x}_i\mathbf{w}_i^\top\gamma)\right\},$$

where for missing y_{i1}:

i) $\hat{\alpha}_{1m}^{(k)} = E(Y_{i1}|Y_{i2} \leq 0) = \hat{\mu}_{i1}^{(k)} - \hat{\rho^*}^{(k)}\lambda(-\hat{\mu}_{i2}^{(k)})$, where $\hat{\mu}_{i1}^{(k)} = \mathbf{x}_i^\top\hat{\boldsymbol{\beta}}^{(k)}$ and $\hat{\mu}_{i2}^{(k)} = \mathbf{w}_i^\top\hat{\gamma}^{(k)}$;

ii) $\hat{\alpha}_{2m}^{(k)} = E(Y_{i2}|Y_{i2} \leq 0) = \hat{\mu}_{i2}^{(k)} - \lambda(-\hat{\mu}_{i2}^{(k)})$;

iii) $\hat{v}_{1m}^{(k)} = E(Y_{i1}^2|Y_{i2} \leq 0) = \hat{\mu}_{i1}^{2(k)} + \hat{\sigma}^{2(k)} - \hat{\rho^*}^{(k)}\lambda(-\hat{\mu}_{i2}^{(k)})(2\hat{\mu}_{i1}^{(k)} - \hat{\rho^*}^{(k)}\hat{\mu}_{i2}^{(k)})$;

iv) $\hat{v}_{2m}^{(k)} = E(Y_{i2}^2|Y_{i2} \leq 0) = 1 + \hat{\mu}_{i2}^{2(k)} - \hat{\mu}_{i2}^{(k)}\lambda(-\hat{\mu}_{i2}^{(k)})$;

v) $\hat{\alpha}_{12m}^{(k)} = E(Y_{i1}Y_{i2}|Y_{i2} \leq 0) = \hat{\mu}_{i1}^{(k)}(\hat{\mu}_{i2}^{(k)} - \lambda(-\hat{\mu}_{i2}^{(k)})) + \hat{\rho^*}^{(k)}$,

and for observed y_{i1}:

vi) $\hat{\alpha}_{2o}^{(k)} = E(Y_{i2}|Y_{i1}, Y_{i2} > 0) = \hat{\mu}_{i2.1}^{(k)} + \sqrt{1 - \hat{\rho}^{2(k)}}\,\lambda\left(\frac{\hat{\mu}_{i2.1}^{(k)}}{\sqrt{1 - \hat{\rho}^{2(k)}}}\right)$;

$\hat{v}_{2o}^{(k)} = E(Y_{i2}^2|Y_{i1}, Y_{i2} > 0) = 1 - \hat{\rho}^{2(k)} + \hat{\mu}_{i2.1}^{2(k)} + \hat{\mu}_{i2.1}^{(k)}\sqrt{1 - \hat{\rho}^{2(k)}}\lambda\left(\frac{\hat{\mu}_{i2.1}^{(k)}}{\sqrt{1 - \hat{\rho}^{2(k)}}}\right)$,

where $\hat{\mu}_{i2.1}^{(k)} = \mathbf{w}_i^\top\gamma^{(k)} + \frac{\hat{\rho}^{(k)}}{\hat{\sigma}^{(k)}}(y_{i1} - \mathbf{x}_i^\top\boldsymbol{\beta}^{(k)})$.

The k-th iteration of the ECM algorithm can be implemented as follows:

E-step:

Given the parameter vector $\boldsymbol{\theta}^* = \hat{\boldsymbol{\theta}}^{*(k)}$, compute the conditional moments $\hat{\alpha}_{1m}^{(k)}, \hat{\alpha}_{2m}^{(k)}, \hat{\upsilon}_{1m}^{(k)}, \hat{\upsilon}_{2m}^{(k)}, \hat{\alpha}_{12m}^{(k)}, \hat{\alpha}_{2o}^{(k)}$, and $\hat{\upsilon}_{2o}^{(k)}$.

CM-steps:

1. Update $\hat{\boldsymbol{\beta}}^{(k)}$ by

$$\begin{aligned}\hat{\boldsymbol{\beta}}^{(k+1)} = (\mathbf{X}^\top \mathbf{X})^{-1} &\left[\hat{\rho}^{*(k)} \left\{ \mathbf{X}^\top \mathbf{W} \hat{\gamma}^{(k)} - \sum_{i=1}^{N_1} \hat{\alpha}_{2o}^{(k)} \mathbf{x}_i - \sum_{i=N_1+1}^{N} \hat{\alpha}_{2m}^{(k)} \mathbf{x}_i \right\} \right. \\ &\left. + \sum_{i=1}^{N_1} \mathbf{y}_{i1} \mathbf{x}_i + \sum_{i=N_1+1}^{N} \hat{\alpha}_{1m}^{(k)} \mathbf{x}_i \right].\end{aligned}$$

2. Update $\hat{\gamma}^{(k)}$ by

$$\begin{aligned}\hat{\gamma}^{(k+1)} = (\mathbf{W}^\top \mathbf{W})^{-1} &\left[\frac{\hat{\rho}^{*(k)}}{\hat{\psi}^{(k)} + (\hat{\rho}^{*(k)})^2} \left\{ \mathbf{W}^\top \mathbf{X} \hat{\boldsymbol{\beta}}^{(k+1)} - \sum_{i=1}^{N_1} y_{i1} \mathbf{w}_i - \sum_{i=N_1+1}^{N} \hat{\alpha}_{1m}^{(k)} \mathbf{w}_i \right\} \right. \\ &\left. + \sum_{i=1}^{N_1} \hat{\alpha}_{2o}^{(k)} \mathbf{w}_i + \sum_{i=N_1+1}^{N} \hat{\alpha}_{2m}^{(k)} \mathbf{w}_i \right].\end{aligned}$$

3. Update $\hat{\psi}^{(k)}$ by

$$\hat{\psi}^{(k+1)} = \exp\left(\hat{\psi}^{*(k+1)}\right) = \frac{1}{N}\left(\textcircled{1} + (\hat{\rho}^{*(k)})^2 \textcircled{2} - 2\hat{\rho}^{*(k)} \textcircled{3}\right),$$

where

$$\textcircled{1} = \sum_{i=1}^{N_1} \left(y_{i1} - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}^{(k+1)}\right)^2 + \sum_{i=N_1+1}^{N} \left(\hat{\upsilon}_{1m}^{(k)} - 2\hat{\alpha}_{1m}^{(k)} \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}^{(k+1)} + \hat{\boldsymbol{\beta}}^{(k+1)^\top} \mathbf{x}_i \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}^{(k+1)}\right),$$

$$\begin{aligned}\textcircled{2} = {} & \hat{\gamma}^{(k+1)^\top} \mathbf{W}^\top \mathbf{W} \hat{\gamma}^{(k+1)} + \sum_{i=1}^{N_1} \left(\hat{\upsilon}_{2o}^{(k)} - 2\hat{\alpha}_{2o}^{(k)} \mathbf{w}_i^\top \hat{\gamma}^{(k+1)}\right) \\ & + \sum_{i=N_1+1}^{N} \left(\hat{\upsilon}_{2m}^{(k)} - 2\hat{\alpha}_{2m}^{(k)} \mathbf{w}_i^\top \hat{\gamma}^{(k+1)}\right),\end{aligned}$$

$$\begin{aligned}\textcircled{3} = {} & \hat{\boldsymbol{\beta}}^{(k+1)^\top} \mathbf{X}^\top \mathbf{W} \hat{\gamma}^{(k+1)^\top} + \sum_{i=1}^{N_1} (y_{i1} \hat{\alpha}_{2o}^{(k)} - y_{i1} \mathbf{w}_i^\top \hat{\gamma}^{(k+1)} - \hat{\alpha}_{2o}^{(k)} \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}^{(k+1)}) \\ & + \sum_{i=N_1+1}^{N} (\hat{\alpha}_{12m}^{(k)} - \hat{\alpha}_{1m}^{(k)} \mathbf{w}_i^\top \hat{\gamma}^{(k+1)} - \hat{\alpha}_{2m}^{(k)} \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}^{(k+1)}).\end{aligned}$$

4. Update $\hat{\rho}^{*(k)}$ by

$$\hat{\rho}^{*(k+1)} = \frac{③}{②}.$$

Therefore,

$$\hat{\sigma}^{2(k+1)} = \exp\left(\hat{\psi}^{*(k+1)}\right) + \left(\hat{\rho}^{*(k+1)}\right)^2, \hat{\rho}^{(k+1)} = \frac{\hat{\rho}^{*(k+1)}}{\sqrt{\exp\left(\hat{\psi}^{*(k+1)}\right) + \left(\hat{\rho}^{*(k+1)}\right)^2}},$$

which will be used for computing the conditional expectations in the next E-step.

3.3. New ECM(NR) algorithm

Based on the `ECM` algorithm mentioned in the previous paragraph, another new estimation algorithm can be derived by combining the Newton-Raphson method, which called `ECM(NR)` algorithm. In the CM-steps of ECM, $\hat{\psi}^{*(k+1)}$ is first updated by the Newton–Raphson method. That is,

$$\hat{\psi}^{*(k+1)} = \hat{\psi}^{*(k)} - \left[\frac{\partial^2}{\partial \psi^{*2}} Q\left(\boldsymbol{\theta}^* | \hat{\boldsymbol{\theta}}^{*(k)}\right)\right]^{-1} \frac{\partial}{\partial \psi^*} Q\left(\boldsymbol{\theta}^* | \hat{\boldsymbol{\theta}}^{*(k)}\right),$$

where

$$\begin{aligned} \frac{\partial}{\partial \psi^*} Q\left(\boldsymbol{\theta}^* | \hat{\boldsymbol{\theta}}^{*(k)}\right) &= \frac{1}{2e^{\psi^*}}\left(-Ne^{\psi^*} + ① + \rho^{*2}② - 2\rho^*③\right) \text{ and} \\ \frac{\partial^2}{\partial \psi^{*2}} Q\left(\boldsymbol{\theta}^* | \hat{\boldsymbol{\theta}}^{*(k)}\right) &= -\frac{1}{2e^{\psi^*}}\left(-Ne^{\psi^*} + ① + \rho^{*2}② - 2\rho^*③\right) - \frac{N}{2}. \end{aligned}$$

Subsequently, $\hat{\psi}^{(k+1)} = \exp\left(\hat{\psi}^{*(k+1)}\right)$.

4. R package for sample selection model

After understanding the basic theory of these algorithms, it is easy to use the Rpackage making process described in chapter 2 to make the package for new estimation methods. First of all, the R package I made is called `NEMpack`, in witch the Metadata: Description and Namespace are as follows, respectively.

- Description

```
Package: NEMpack
Type: Package
Title: Some new estiamtion methods for a Heckman Selection Model
Version: 0.1.0
Author: Yang Kexuan
Maintainer: Yang Kexuan <kexuanY@163.com>
Description: This package is for the simple description of
how to make R packages.
Depends: R (>= 2.10)
License: GPL-2
Encoding: UTF-8
LazyData: true
RoxygenNote: 6.1.1
```

- Namespace

```
# Generated by roxygen2: do not edit by hand
S3method(print,ECM)
S3method(print,ECMnr)
export(ECM)
export(ECMnr)
importFrom(sampleSelection,selection)
```

Source code about these two new algorithms and Documentation are given in Appendices A and B, respectively.

For the sake of conciseness, I will take the first algorithm `ECM` as an example to introduce the R source code and the Documentation. In this case, Smoke is used as my dataset, this dataset is a subset of data used in Mullahy (1987). Data was collected through the Smoking Supplement to the US National Health Interview Survey in 1979 and 1980. The dataset contains 807 cross-sectional individual data on smoking, and it contains the following columns:

1. educ: years of schooling.

2. age: respondents age in years.

3. cigpric: state cigarette price, cents per pack.
4. income: annual income in USD.
5. restaurn: dummy variable indicating if state restaurant smoking restrictions are in place.
6. smoker: dummy variable indicating if person has smoked at least one cigarette.
7. cigs_intervals: number of cigarettes smoked per day, coded in intervals with intervals boundaries: (0,5,10,20,50)
8. cigs: number of cigarettes smoked per day.

In this example, `cigs_intervals` and `educ` create a response regression formula, and in the selection formula, `Smoker` was used as a response variable, `age` and `educ` are used as independent variables. And the estimation results obtained in the console:

```
$‘beta‘
[,1]
[1,] 2.2867659
[2,] 0.0355655

$gamma
[,1]
[1,]  0.93556065
[2,] -0.06438668
[3,] -0.01063360

$sigma
[1] 0.9901125

$rho
[,1]
[1,] 0.3933366
```

Although the estimation results seem to have no problem, there is no user interface that is easy for first-time users to understand. Moreover, I hope that other users can use the estimation method in this package to analyze any data he wants to study, so here I need to improve the function part. In Leisch (2008), there is an introduction to how to make the user interface more beautiful and tidy, and based on it, I will define classes for my `ECM` example.

Firstly, I will define `ECM` function mentioned in the previous section. In fact, the main part of the function is the same as the original code, except that we need to add some code for data import and beautification results interface.

```
#' @export
ECM <- function(response,selection,data, method="original",
eps = 10^(-10)){
cl <- match.call()
mf <- match.call(expand.dots = FALSE)

m1 <- match(c("response", "res.var"), names(mf), 0)
mf1 <- mf[c(1, m1)]
mf1$drop.unused.levels <- TRUE
mf1$na.action <- na.pass
mf1[[1]] <- quote(model.frame)
names(mf1)[2] <- "formula"
mf1 <- eval(mf1, parent.frame())
mt1 <- attr(mf1, "terms")

m2 <- match(c("selection", "data"), names(mf), 0)
mf2 <- mf[c(1, m2)]
mf2$drop.unused.levels <- TRUE
mf2$na.action <- na.pass
names(mf2)[2] <- "formula"
mf2[[1]] <- quote(model.frame)
mf2 <- eval(mf2, parent.frame())
mt2 <- attr(mf2, "terms")

y1<- model.response(mf1, "numeric")
y2<- model.response(mf2, "numeric")

x <- model.matrix(mt1, mf1, contrasts.arg = NULL, xlev = NULL)
w <- model.matrix(mt2, mf2, contrasts.arg = NULL, xlev = NULL)
```

```
x.name <- names(as.data.frame(x))
w.name <- names(as.data.frame(w))
...
```

It can be seen that compared to the original function code, some new parameter definitions have been added. The newly added elements are prepared for importing data and beautifying the user interface. You can refer to the `lm` and `selection` functions in R and on CRAN for their respective definitions of functions, and you will find that they are so similar about this part. (Because they are the source code for efficient calculations in the R function. For example, in my case I want to define response equation and selection equation, respectively, it can be done by using the $match.call()$ function, which is used to record the call for later re-use.)

Then call my function for parameter estimation, so I need to add estimated values of four parameters to the results. Finally, setting the class of the return object to "`ECM`". They can be done by using the `print()` method to beautify my results interface.

```
#' @method print ECM
#' @export
print.ECM <- function(object, digits = max(3, getOption("digits") - 3))
{
cat("\nCall:\n")
print(object$call)
cat("\nResponse coefficients:\n")
print(t(data.frame(Estimate = object$estimate_response,
row.names = object$names_response, check.names=FALSE)), digits = digits )
...

cat("\n")
}
```

Then the final result is as follows:

```
> print.ECM(a)
```

```
Call:
ECM(response = Smoke$cigs_intervals ~ Smoke$educ, selection = Smoke$smoker ~
Smoke$educ + Smoke$age, data = Smoke)

Response coefficients:
(Intercept) Smoke$educ
Estimate       2.287    0.03557

Selection coefficients:
(Intercept) Smoke$educ Smoke$age
Estimate      0.9356   -0.06439  -0.01063

Sigma:
(sigma)
Estimate  0.9901

Rho:
(rho)
Estimate 0.3933
```

Next, I want to add the summary function to put the estimated value and the standard error analysis result together in the result interface. Firstly, define the summary function:

```
#' @method summary ECM
#' @export
summary.ECM <- function(object,...)
{
z.value_response <-
as.vector(object$estimate_response)/as.vector(
object$std_error_response)
...

)

result$names_response <- object$names_response
...
```

```
class(result) <- "summary.ECM"
result
}

#’ @rdname summary.ECM
```

Then, define the print function for summary function as well to view the input results:

```
#’ @method print summary.ECM
#’ @export
print.summary.ECM <- function(object, digits = max(3,
getOption("digits") - 3))
{
cat("\nCall:\n")
print(object$call)
cat("\n--------------- Estimations and  Standard Error Results ----------\n")
cat("\nResponse equation:\n")
out <- data.frame( estimate = object$estimate_response,
std.error = object$std_error_response, z.value = object$z.value_response,
pval = 2*pnorm(abs(object$z.value_response), lower.tail = F) )
Signif.out <- symnum(out$pval, corr = FALSE, na = FALSE,
cutpoints = c(0, 0.001, 0.01, 0.05, 0.1, 1),
symbols = c("***", "**", "*", ".", " "))
out <- cbind(out, format(Signif.out))
row.names(out) <- object$names_response ; colnames(out) <- c("Estimate",
"Std. Error", "Z Value", "Pr(>|Z|)", "")
print(out, digits = digits)

...
cat("---\n")
cat("Signif. codes:", 0, "’***’", 0.001, "’**’", 0.01,
"’*’", 0.05, "’.’", 0.1, "’ ’", 1, "\n\n")

}
```

Also by analyzing the data from Smoke dataset, the analysis result of the summary function is as follows:

```
Call:
ECM(response = Smoke$cigs_intervals ~ Smoke$educ, selection = Smoke$smoker ~
Smoke$educ + Smoke$age, data = Smoke)

-------------- Estimations and  Standard Error Results ---------------

Response equation:
Estimate Std. Error Z Value  Pr(>|Z|)
(Intercept)  2.28677    0.28050   8.152 3.567e-16 ***
Smoke$educ   0.03557    0.02235   1.592 1.115e-01


...
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It can be seen that we can assign values to the parameters and add the data you want to analyze directly in the `ECM` function, and the result interface are also more detailed and easier to understand than before.

Also, I made the function `coef`, it is simple and also looks same as `lm` and `glm` cases, so I exclude it on this case. Making functions and documentations for this package is done, but there is still a lot to learn about the process and method of making R packages. And I will continue to learn more efficient and convenient ways of making this package, while checking it in order to submit it to the CRAN.

Chapter 4. Conclusions

In this thesis, I have illustrated how to make an R package in detail by using a combination of graphics and text in the second chapter. Whether it is a single function or multiple functions, we can easily turn it or them into a regular R package, which can easily record the original code and documentation of your function. It is not only for your own use but also for checking and distributing to other R users.

In the third chapter of this thesis, I have made an R package for some new EM-type algorithms developed by Zhao et al. (2019) for the Heckman selection model. And in this package I added the "print.ECM" function and the "summary.ECM" function to have more detailed results which are similar to the "lm" or "glm" function. As an introductory tutorial, the R package system has many features that are not mentioned in this thesis, but the basic package making process is included here for beginners.

References

Arellano-Valle, R.B., Branco, M.D. and Genton, M.G. (2006) "A unified view on skewed distributions arising from selections," *The Canadian Journal of Statistics*, 34, 581-601.

Bates, D. (2019), Introduction to the Matrix package—as of Feb. 2005, *Matrix: Sparse and Dense Matrix Classes and Methods*, https://CRAN.R-project.org/package=Matrix.

Heckman, J. (1974) "Shadow prices, market wages, and labor supply," *Econometrica*, 42, 679-694.

Heckman, J. (1979), "Sample selection bias as a specification error," *Econometrica*, 47, 153-161.

Leisch, F (2008), Creating r packages: A tutorial, *Technical report*, Physica-Verlag.

Little, R.J.A. and Rubin, D.B. (2002), *Statistical analysis with missing data, 2nd edition*, New Jersey: Wiley.

Meng, X.-L., Rubin, D. B. (1993), Maximum likelihood estimation via the ECM algorithm: A general framework, *Biometrika*, Oxford University Press.

Mullahy, J. (1987), Cigarette smoking: Habits, Health concerns, and Heterogenous Unobservables in a Microeconometric Analysis of Consumer Demand, *Cigarette Smoking*.

R core team (1999), Writing R extensions, *R Foundation for Statistical Computing*, https://cran.r-project.org/doc/manuals/r-release/R-exts.html.

Weast, R. C., ed. (1973), *Handbook of chemistry and physics*, CRC Press.

Wickham, H. (2015), *R packages*, O'Reilly Media, Inc.

Wickham, H. (2016), *ggplot2: Elegant Graphics for Data Analysis*, Springer-Verlag New York.

Zhao, J., Kim, H.-J. and Kim, H.-M. (2019), New EM-type algorithms for the Heckman selection model, *submitted*.

Abstract(in Korean)

R 패키지 만들기

양가선
응용통계학과
건국대학교 대학원

이 논문에서는 R를 배운 초보자를 위해 R 패키지를 만드는 방법에 대해 자세히 설명한다. 여러 가지 간단한 기능부터 시작하여 R 패키지를 만드는 전체적인 과정을 소개한다. R 패키지를 만드는 방법, 메타 데이터 및 패키지를 만들고 다른 사람들과 공유하는 방법에 대해 설명한다. 또한 Heckman 표본 선택 모형을 위해 EM 알고리즘을 기반으로 Zhao et al. (2019) 에서 개발된 새로운 추정 방법을 이용하였다. R 패키지를 만드는 방법과 결합하여 이러한 알고리즘을 사용하고 다른 R 사용자와 공유하기 위해 패키지를 만들었다.

주제어 : R 패키지, 표본 선택, Heckman 모형, EM 알고리즘.