Integration of Reusable C++ Code and Mathematical Libraries in R Packages with Rcpp

Part 1

CFRM 524: Advanced C++ for Finance

Daniel Hanson

The Problem

- We are developing a mathematical model in R
- Due to R being an interpreted language, some of our code will probably take a long time to run
- C++ is an answer for higher performing code
- We also wish to ensure that our C++ code is standard and portable
  - Code that we use from a standard/reusable C++ code base
  - Code that we write (that can go into the standard C++ code base)

The R Package Solution

- The Rcpp R package* provides a reasonably painless means of creating an interface from R to C++
- Using the RStudio IDE with the GNU gcc compiler, we can write or import C++ code, compile it, and call it from R
- The code files are stored and managed in an RStudio project, much like integrated code in other popular IDE's such as Visual Studio
- Popular and powerful open source mathematical C++ libraries such as Boost and Eigen can also be integrated without excessive wailing or gnashing of teeth
- We can build the solution into an R package
  - Compile and build it once
  - Deploy it on an arbitrary number of machines
  - Use the built-in documentation tools

---

*Package author: Dirk Eddelbeuttel, http://dirk.eddelbuettel.com/code/rcpp.html

RStudio IDE

RcppPackagePart01 - RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function | Addins | RcppPackagePart01

CppInterface.cpp | temp003a.R

Source on Save | Source

```
1  #include "ReusableCppSourceCode.h"
2  #include <vector>
3  #include <cmath>
4  #include <algorithm>
5  #include <Rcpp.h>     // For Rcpp::NumericVector
6
7  using std::vector;
8  using std::copy;
9  using Rcpp::NumericVector;
10
11 // Declaration of helper function
12 vector<double> rcppToStdVec(const Rcpp::NumericVector& nv);
13
14 // Nonmember Function Interfaces:
15
16 // [[Rcpp::export]]
17 int rcppAdd(double x, double y)
18 {
19   return add(x, y);
20 }
21
22 // [[Rcpp::export]]
23 NumericVector rcppSortVec(NumericVector v)
24 {
25   vector<double> stlVec(v.size());
26   copy(v.begin(), v.end(), stlVec.begin());
27
28   // Call the reusable sortVec(.) function:
29   stlVec = sortVec(stlVec);
30
31   // Copy results in vector<int> back into NumericVector.
32   // Use same vector v since it has the same length:
33   copy(stlVec.begin(), stlVec.end(), v.begin());
34
35   // Return as rcpp::NumericVector:
36   return v;
37 }
```

14:20 (Top Level) C/C++

Console | Terminal | Jobs

C:/Users/djhanson/OneDrive/Surface/CFRM/CFRM524/2020/SourceCode/Rcpp/RcppPackagePart01/

```
+   layout(title="Areas of Squares")
>
```

Environment | History | Connections | Build | Tutorial

Import Dataset | List

Global Environment

Data	
dfSq	10 obs. of 2 variables
Values	
circleAreas	num [1:10] 3.14 12.57 28.27 50.27…
circleRadii	int [1:10] 1 2 3 4 5 6 7 8 9 10
squareAreas	num [1:10] 1 4 9 16 25 36 49 64 8…
squareSides	int [1:10] 1 2 3 4 5 6 7 8 9 10
v	num [1:1000] -0.5022 0.1315 -0.07…
x	int [1:5] 5 4 3 2 1

Files | Plots | Packages | Help | Viewer

New Folder | Delete | Rename | More

FRM > CFRM524 > 2020 > SourceCode > Rcpp > RcppPackagePart01

Name	Size	Modified
..		
.Rbuildignore	30 B	May 29, 2020, 3
DESCRIPTION	357 B	May 29, 2020, 3
man		
NAMESPACE	109 B	May 29, 2020, 4
R		
RcppPackagePart01.Rproj	320 B	May 29, 2020, 5
Read-and-delete-me	430 B	May 29, 2020, 3
src		

RStudio IDE

InterfaceRcppProject - RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

moreCppExamples.R | CppInterface.cpp | ReusableCppSourceCode.cpp

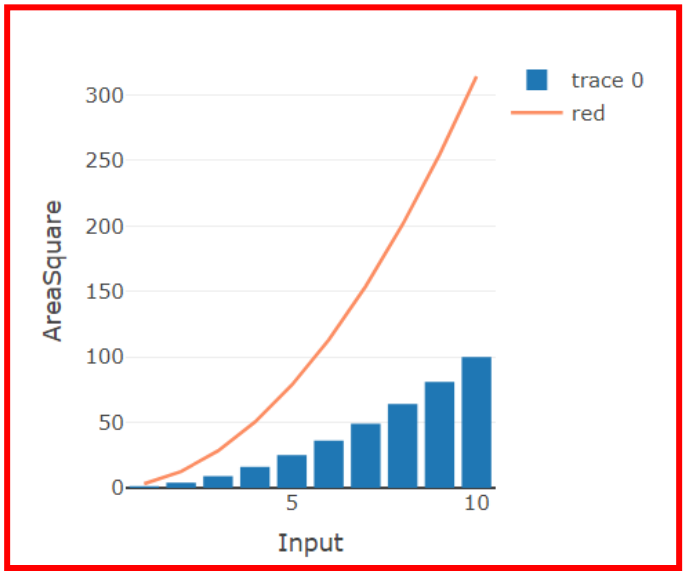
```
## Example 7:
# Create R vectors containing constructor parameters
# for both the Square and Circle classes.  Then, use
# sapply in R to vectorize each area calculation.
# Finally, plot the results in R, showing that we can
# take results computed in C++, and use them in
# R visualization tools that don't exist in C++:
squareSides <- c(1:10)
circleRadii <- c(1:10)

squareAreas <- sapply(squareSides, squareArea)
circleAreas <- sapply(circleRadii, circleArea)

### Using plotly
library(plotly)
dfSq <- matrix(data=c(squareSides, squareAreas), nrow=length(squareSides),
dfSq <- as.data.frame(dfSq)
colnames(dfSq) = c("Side", "Area")
rownames(dfSq) = NULL
plot_ly(dfSq, x = ~Side, y = ~ Area, type = "bar")


dfCirc <- matrix(data=c(circleRadii, circleAreas), nrow=length(squareSides)
dfCirc <- as.data.frame(dfCirc)
```

Environment | History | Connections | Tutorial

Global Environment

apiSortVec	function (v)
apiStdMean	function (vec)
circleArea	function (radius)
rcppAdd	function (x, y)
rcppIntegra...	function (x)
rcppSortVec	function (v)
rcppStdMean	function (vec)
squareArea	function (side)

Files | Plots | Packages | Help | Viewer



Console | Terminal | Jobs

C:/Users/djhanson/OneDrive/Surface/CFRM/CFRM524/2020/SourceCode/Rcpp/InterfaceRcppProject/

The Agenda

- Construct and generate an R package containing reusable C++ nonmember and class member functions, called via an interface layer in C++, exposing these interface functions in R (Part 1)

- Generate documentation for R functions exported by the Rcpp interface layer (extending our Part 1 discussion)

- Use external mathematical C++ libraries (Part 2)
  - Eigen Matrix Algebra Library (RcppEigen package)
  - Boost Libraries (add the BH package to the project)
  - There could be others (if supported by Rcpp)

Preliminary Rcpp Setup and Configuration

Integrating the gcc C++ compiler and Rcpp

Setup

- We will design our integrated R package in an RStudio project

- Below, we will go through the process
  - R
  - RStudio
  - C++ compiler

Setup: R and RStudio

- First, make sure you have the latest version of R installed (4.0.0)

- Next, make sure you have the latest version of RStudio Desktop installed: RStudio v1.3.959
- It may be downloaded here:

  https://rstudio.com/products/rstudio/download/preview/

- This version includes a bug fix related to displaying graphs using the ***plotly*** R package

Setup: C++ Compiler and Rtools

- We will need a modern C++ compiler
- Rcpp requires the gcc (or Clang) compiler
- It will not work with the Microsoft Visual Studio compiler
- On Windows 10: Download and install Rtools:
  - Download executable and follow the directions on
    https://cran.r-project.org/bin/windows/Rtools/
    - On Windows 64-bit: rtools40-x86_64.exe
    - On Windows 32-bit: rtools40-i686.exe (i386 compilers only)
  - What is this thing called Rtools?
    - Installs the gcc 8.3.0 C++ compiler on your machine
    - Allows you to
      - Use Rcpp and build your own R packages with integrated C++ code
      - Download packages from source and build them locally
    - Again, Rcpp is NOT compatible with Visual Studio
    - It requires a gcc compiler for Windows

C++17 Limitations with Rtools and gcc 8.3.0

- The availability of modern C++17/C++20 language and library features will depend upon the version of the gcc compiler you use
- Rtools installs MinGW-64 ("Minimalist GNU for Windows"), which contains the gcc 8.3.0 C++ compiler (Feb 2019)
- It contains some, but not all, C++17 language and library features
  - Contains:
    - Special math functions (Bessel functions, Legendre polynomials, etc)
    - **`std::variant`**, **`std::optional`**, **`std::any`**
  - Unfortunately does not contain parallel STL algorithms
  - See https://gcc.gnu.org/onlinedocs/gcc-8.3.0/libstdc++/manual/manual/status.html#status.iso.2017
- For this demonstration, we will use this platform
- Latest stable gcc compiler version: gcc 9.3
  - Mac (or use the most recent Clang compiler in Xcode)
  - Linux
  - Windows (but without the convenience of Rtools)

Setup: C++ Compiler and Rtools

- Once R and your compiler (Rtools) have been installed and configured:
  - Update the **`Makeconf`** file in your R installation (this is not in the instructions on the Rtools website)
    - Make a copy of **`.../R-4.0.0/etc/x64/Makeconf`** (eg, **`Makeconf.bak`**)
    - Open your Makeconf file with a text editor (eg Notepad++), and locate the line

      ```
      CXX = $(BINPREF)g++ -std=gnu++11 $(M_ARCH)
      ```

    - Change this to

      ```
      CXX = $(BINPREF)g++ -std=gnu++17 $(M_ARCH)
      ```

    - And then, save and close the file
    - The reasons for this will become clearer when we discuss C++17 with Rcpp

Setup: C++ Compiler and Rtools

- Modifying the **Makeconf** file (continued):

```
CFLAGS = -O2 -Wall $(DEBUGFLAG) -std=gnu99 -mfpmath=sse -msse2 -mstackrealign
CPICFLAGS =
CPPFLAGS =
CXX = $(BINPREF)g++ -std=gnu++11 $(M_ARCH)
CXXCPP = $(CXX) -E
CXXFLAGS = -O2 -Wall $(DEBUGFLAG) -mfpmath=sse -msse2 -mstackrealign
```

```
CFLAGS = -O2 -Wall $(DEBUGFLAG) -std=gnu99 -mfpmath=sse -msse2 -mstackrealign
CPICFLAGS =
CPPFLAGS =
CXX = $(BINPREF)g++ -std=gnu++17 $(M_ARCH)
CXXCPP = $(CXX) -E
CXXFLAGS = -O2 -Wall $(DEBUGFLAG) -mfpmath=sse -msse2 -mstackrealign
```

Setup on Mac: R/RStudio/C++ Compiler

- For Mac Installation:
  - First, make sure you have the latest version of R installed (4.0.0)
  - You do not need to install Rtools on the Mac
  - You will need to install Xcode (installs the Clang compiler):

    Execute the command **`xcode-select --install`** in a Terminal session

    ➢ Additional information here: https://teuder.github.io/rcpp4everyone_en/020_install.html
  - It is possible that you will need to remove earlier versions of the Xcode/Clang compiler; see this URL for reference (but don't worry about the Fortran stuff): https://thecoatlessprofessor.com/programming/cpp/r-compiler-tools-for-rcpp-on-macos/
  - You will presumably need to modify the Makeconf file in your R installation, as described in the directions above (if not set for C++17)

Setup: Installation of Rcpp

- Now, open RStudio, and install **`Rcpp`**
  - Either run the following R command:

```
install.packages("Rcpp")
```

  - Or, install using the RStudio Tools/Install Packages menu selection

- Also install the **`RcppEigen`** and **`BH`** packages

Building an R Package with Rcpp
Case 1: No External Libraries

Integrate Reusable C++ Code in an R Package

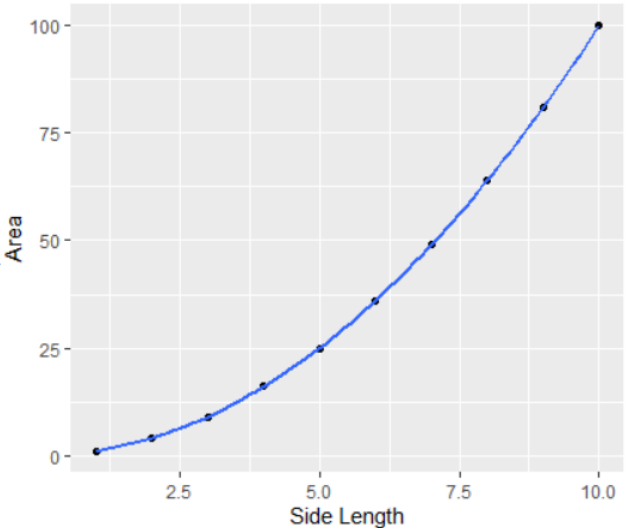
Using Rcpp

Case w/o external C++ libraries

- We wish to use functions in a Standard C++ code base, called from R
- R users will not see or need to care about the C++ code

C++ functions exposed in R and R graphing functions

```r
squareSides <- c(1:10)
circleRadii <- c(1:10)

squareAreas <- sapply(squareSides, squareArea)
circleAreas <- sapply(circleRadii, circleArea)

### Using qplot in ggplot2:
library(ggplot2)

dfSq <- matrix(data=c(squareSides, squareAreas), nrow=length(squareSides), ncol=2)
dfSq <- as.data.frame(dfSq)

dfCirc <- matrix(data=c(circleRadii, circleAreas), nrow=length(squareSides), ncol=2)
dfCirc <- as.data.frame(dfCirc)
colnames(dfCirc) = c("Radius", "Area")

# qplot
qplot(x = squareSides, y = squareAreas, data =dfSq,
      geom = c("point", "smooth"),
      main = "Area of Square vs Side Length",
      xlab = 'Side Length', ylab = 'Area')

qplot(x = Radius, y = Area, data =dfCirc,
      geom = c("point", "smooth"),
      main = "Area of Circle vs Radius Length",
      xlab = 'Radius Length', ylab = 'Area',
      colour = I("darkred"))
```



C++ Interface Layer

Non-member interface funtions

Rcpp tags

Independent of reusable C++ code

Must use Rcpp library to handle conversion of R vectors to std::vector

Financial models code base

Standard C++ ONLY

classes

non-member functions

C++ Standard Library (gcc 8.3.0)

Converting R vectors to `std::vector<.>` in C++

- We cannot, in general, pass a real R vector to a C++ function as a **`std::vector<double>`** object

- The Rcpp package contains a variety of functions and classes that facilitate the interface between R and C++
  - **`#include<Rcpp.h>`**
  - **`Rcpp`** namespace

- For now, we only need one class from Rcpp:
  - **`Rcpp::NumericVector`**
    - An STL-compliant container class that handles input from and output to an R numeric vector
    - Has several member functions with the same name and purpose as those on **`std::vector<.>`**, including **`size()`**, which is convenient for us
    - Can copy to an **`std::vector`** in the interface using **`std::copy(.)`**

Converting R vectors to `std::vector<.>` in C++

- Interface function exported to R is indicated by the tag above its signature:

```
// [[Rcpp::export]]
double fcn(Rcpp::NumericVector v)
{
    std::vector<double> w(v.size());
    std::copy(v.begin(), v.end(), w.begin());
    // Call function in standard C++ code base:
    double y = doSomething(w);

    return y;
}
```

C++ Interface Layer
Non-member interface funtions
Rcpp tags
Independent of reusable C++ code
Must use Rcpp library to handle conversion of R vectors to std::vector

Financial models code base
Standard C++ ONLY

classes
non-member functions

- **NOTE:** ***This should only be done at the interface level --*** Do ***not*** use the **Rcpp** declaration or namespace in the standard C++ code base

Case w/o external C++ libraries

- *Prime Directive*: Keep reusable code ***standard*** and ***independent*** of Rcpp interface
- We need to use the **`NumericVector`** types from the Rcpp namespace

R Functions

```
rcppAdd(x, y)  # x, y numeric
rcppSortVec(v)  # R vector v
rcppStdMean(v)  # R vector v
squareArea(x)  # x = side
squareCircle(r) # r = radius
```

C++/Rcpp Interface Functions

```
int rcppAdd(double x, double y)
Rcpp::NumericVector rcppSortVec(Rcpp::NumericVector v)
double rcppStdMean(Rcpp::NumericVector v)
double squareArea(double side)
double circleArea(double radius)
```

Reusable C++ Code

C++ Functions

```
double add(double x, double y)
vector<double> sortVec(vector<double> v)
double stdMean(vector<double> vec)
```

C++ Class Square

```
Square(double side)
double area() const
```

C++ Class Circle

```
Circle(double radius)
double area() const
```

C++17 Example

- We will also use the following special math functions from C++17 in a UDF called **`integralSum(.)`**:
  - Elliptic integral of 1st kind **`std::ellint_1(double k, double phi)`**
    - $\int_0^{\varphi} \frac{d\theta}{\sqrt{1-k^2 sin^2\theta}}$
  - Exponential integral **`std::expint(double x)`**
    - $\int_x^{\infty} \frac{e^{-t}}{t} dt$

R Function

```
rcppIntegralSum(k, phi, x)
```

C++/Rcpp Interface Function

```
double rcppIntegralSum(double k, double phi, double x)
```

Reusable C++ Code

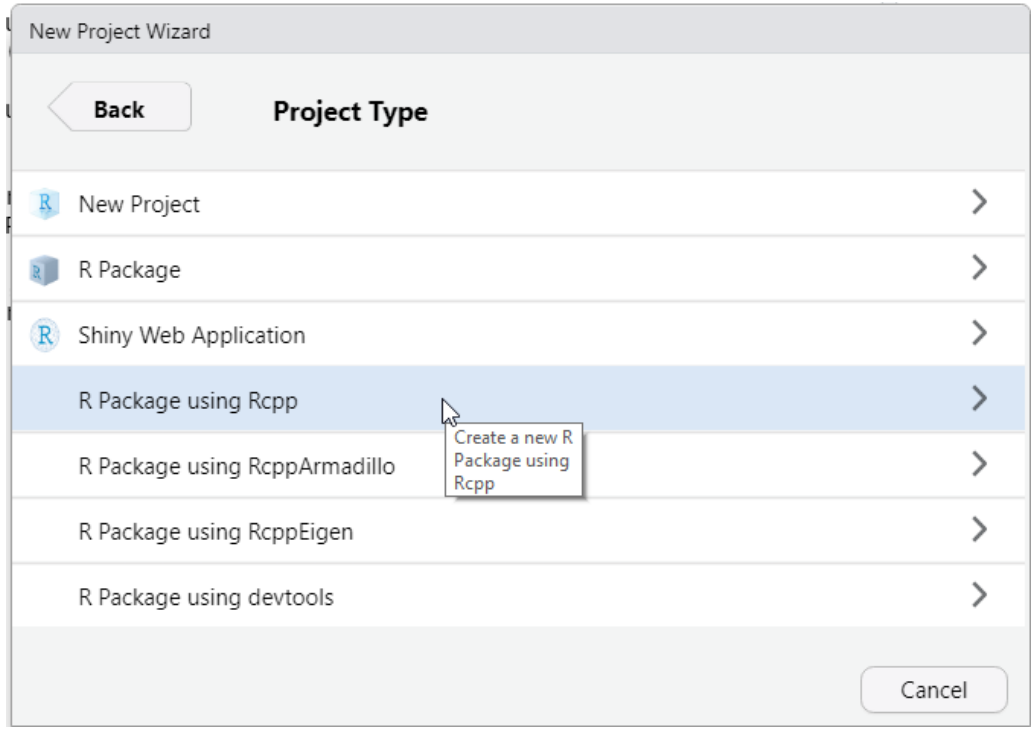
C++ Function

```
double integralSum(double k, double phi, double x)
```

$$\int_0^{\varphi} \frac{d\theta}{\sqrt{1-k^2 sin^2\theta}} + \int_x^{\infty} \frac{e^{-t}}{t} dt$$

Creating an Rcpp Package Project in RStudio

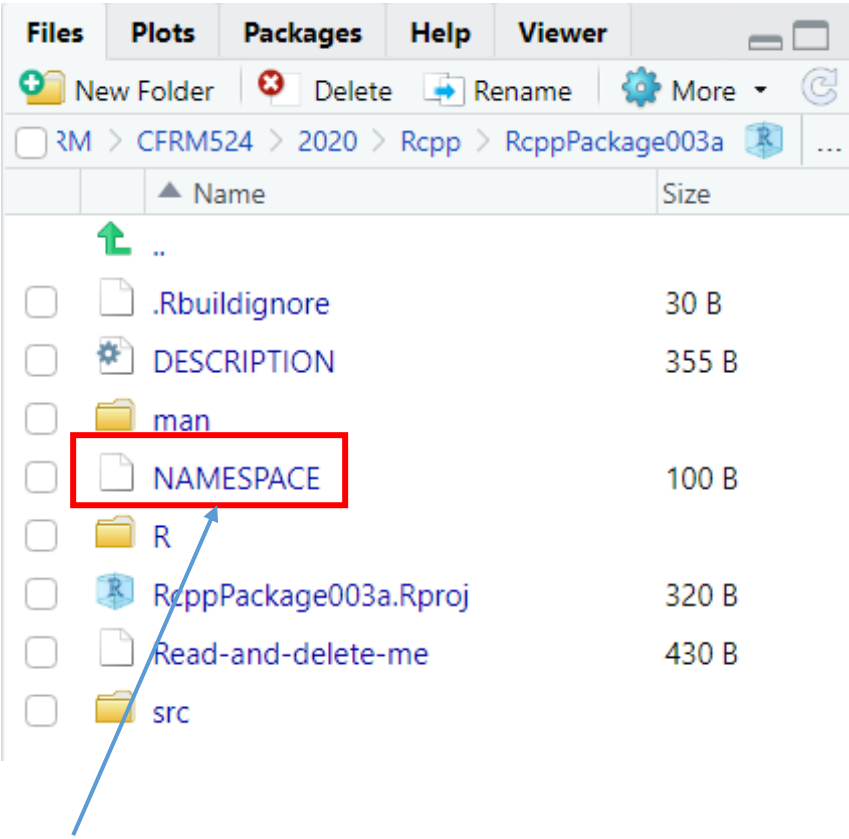
- We will look at this shortly in an example, but here is an overview:
  - Select ***File*/*New Project*/*New Directory*/*R Package*** using Rcpp



  - Enter the directory path and new subdirectory name, and create the project; the subdirectory will be the name of your R package

Creating an Rcpp Package Project in RStudio

- In the Files pane in RStudio
  - **man**: Documentation folder ("manual")
  - **R**: R code in the package (we will not have any, but it is completely legal)
  - **src**: C++ code in the package



- Click on the NAMESPACE file to open it:

- In the **NAMESPACE** file

Read-and-delete-me × NAMESPACE ×

```
1  useDynLib(RcppPackage003a, .registration=TRUE)
2  export("Rcpp.fake.fun")
3  importFrom(Rcpp, evalCpp)
4
```

- Delete line 2: **`export("Rcpp.fake.fun")`**
- Append the line **`exportPattern("^[[:alpha:]]+")`**
- Save the file

Read-and-delete-me × NAMESPACE ×

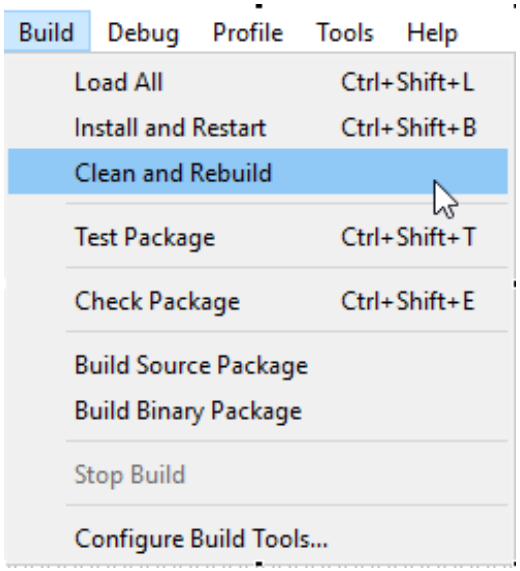
```
1  useDynLib(RcppPackage003a, .registration=TRUE)
2  importFrom(Rcpp, evalCpp)
3  exportPattern("^[[:alpha:]]+")
4
```

Example: Rcpp Package with Reusable C++ Code

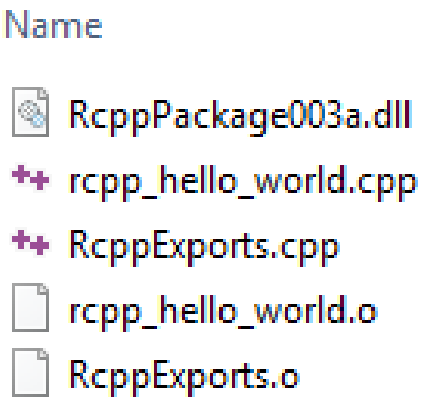
- We will now look at an example of building an R package with Rcpp with reusable C++ code (as diagrammed earlier)
- Remember: At the interface level, we need to convert between **`Rcpp::NumericVector`** and **`std::vector<double>`**
- Complete code can be found in the **`RcppPackagePart01`** RStudio project
- We will construct, build, and show how to deploy the example package
- We will create it using the name **`RcppTemp01`** in the following step-by-step example package, so as not to conflict with the completed package (**`RcppPackagePart01`**)
  - Create an Rcpp R package project in RStudio
  - Hello World
  - Build up the example as shown in the earlier R/Interface/Implementation diagram

Example: Building the Rcpp Package Project in RStudio

- Build the package by selecting from the ***Build*** menu



- ➢ This will compile the C++ code and export the interface functions to R
- ➢ For the simple Hello World case, note the .o files for each .cpp file have been generated, along with the shared library (dll)
- ➢ This can be checked in Windows File Explorer (src):



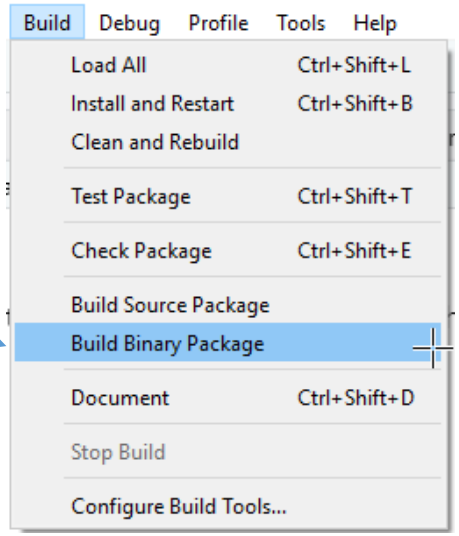
Example in RStudio: Interface Files

- Setting up the C++ interface files:
  - **`#include`** the declaration files for the reusable code base
    - The two classes (Square and Circle) in the CppInterface2.cpp file
    - Everything else in CppInterface.cpp
  - Can think of **`CppInterface.cpp`** or **`CppInterface2.cpp`** as a file that would contain **`main()`** in a typical C++ executable project (no declaration file)
  - Indicate each interface function to be callable in R with the `// [[Rcpp::export]]` tag

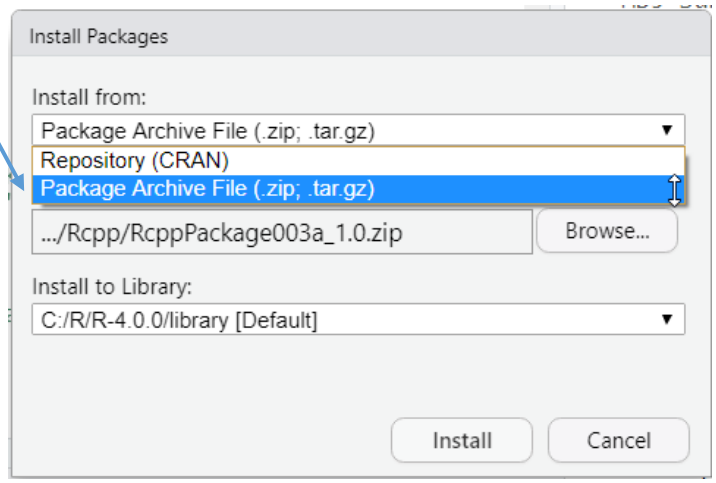
- Let's now look at this first example

Distributing the Package

- To distribute the package as a binary
  - Select Build Binary Package from the Build options



  - This will generate
    - **`RcppPackagePart01.zip`** on Windows
    - **`RcppPackagePart01.tar.gz`** on the Mac (and Linux)
  - Located in directory one above the project directory
  - Copy to/download on a different machine
  - Open up a new RStudio session on this other machine
    - Install the package as a local archive file
    - Put **`library(RcppPackagePart01)`** in the local R session to load it
    - Call the functions



Example/Documentation

- We can also generate standard CRAN-compliant documentation for the package using RStudio utilities

rcppAdd {RcppPackage003a} R Documentation

A function that adds two integers and returns the sum

Description

This function has no social value

Usage

```
rcppAdd(x, y)
```

Arguments

`x, y`	integers

Value

Returns the sum of the two integers

[Package *RcppPackage003a* version 1.0 Index]

- This will be covered in the next presentation deck

Summary

- We have now seen how to create a hybrid R package with a front-end in R calling C++ code, reasonably painlessly
  - Again, reusable C++ that requires no modifications
  - The C++ interface
    - May be contained in a single .cpp file
    - Or, in multiple .cpp files
    - Contains Rcpp-specific directives and **`#include`** statements
    - And again, results returned by these interface functions may be used in powerful R graphical tools, as well as in any R function
  - In addition, we can add our own R code to our package
    - Graphing functions
    - Other functions to be exported to the end user

- Next (Part 1b): We will cover documentation for the R functions in our package: The exported the C++ functions from the interface layer
- And then, (Part 2): We will import open source math libraries
  - Eigen
  - Boost

[END – Part 1]