Integration of Reusable C++ Code in R Packages with Rcpp

Northwest C++ Users Group

2021.01.20

Daniel Hanson

The R Language
R Packages
RStudio IDE

The R Language

- The R Project for Statistical Computing
  - A free software environment for statistical computing and graphics
  - Also contains a convenient and robust set of linear algebra functions
  - Runs on a wide variety of UNIX/Linux platforms, Windows, and MacOS
- An interpreted language; can be run
  - Interactively
  - As a program
- Syntax similar to C++ (keywords, code blocks in braces, etc)
- Built-in math/statistical functions
  - Compiled in C/C++/FORTRAN
  - Reasonably fast if vectorized
- Supports functional and object-oriented programming
- Can’t vectorize everything
  - Limitations of an interpreted language
  - Code that takes days to run can sometimes be reduced to minutes in C++

Some R Code

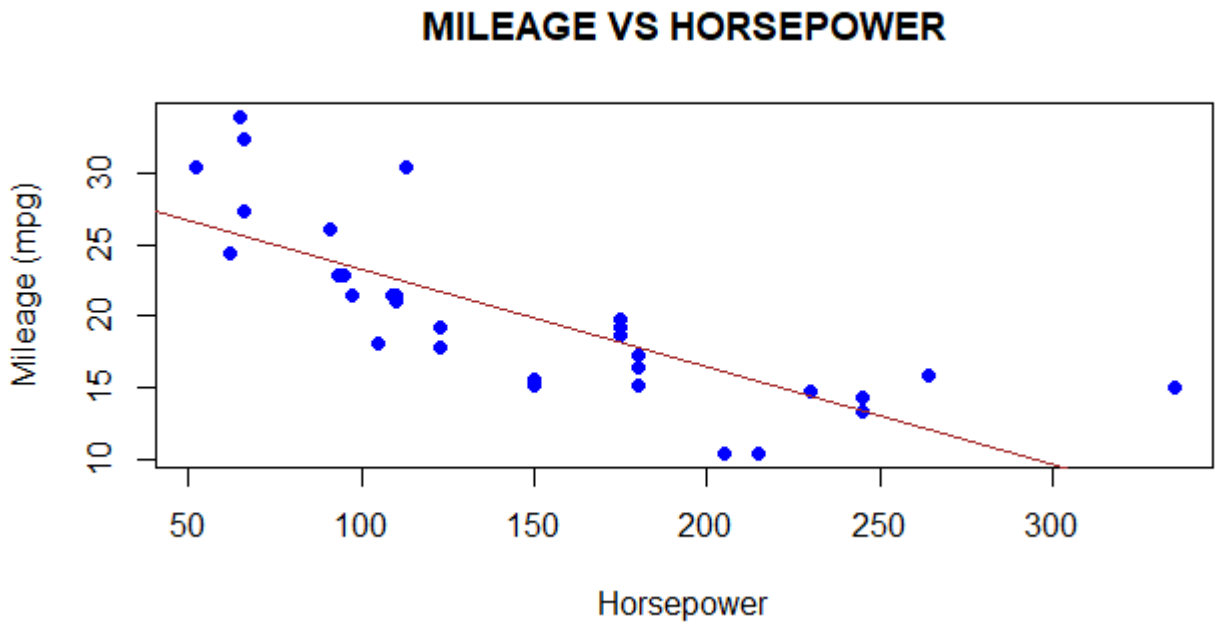
```
trigFcn <- function(x)
{
  y <- 1.06 * x
  sin(y) + cos(y)
}

multiTrig <- function(x, y, z)
{
  return(sin(x) + cos(y) + tan(z))
}

# Source these functions and then call them interactively:
trigFcn(-5.6)
multiTrig(0.1, 0.2, 0.5)
# Alternatively, can explicitly put the argument names in:
multiTrig(y = 0.2, z = 0.5, x = 0.1)
```

More R Code (Linear Regression)

```
attach(mtcars)    # dataframe

# Simple linear regression: Regress mpg on hp:
fit <- lm(mpg ~ hp, data = mtcars)

# Overall summary:
summ <- summary(fit)
summ$coefficients   # Note we get t-test results here
summ$adj.r.squared
summ$sigma

# Plot the data points with regression line:
plot(x = hp, y = mpg, pch = 16, cex = 1.0, col = "blue",
     main = "MILEAGE VS HORSEPOWER", xlab = "Horsepower",
     ylab = "Mileage (mpg)")
# Overlay regression line; just put in lm object 'fit' as argument:
abline(reg = fit, col = "brown")    # reg = regression object
```

Results from regression example:

```
> summ$coefficients   # Note we get t-test results here
               Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 30.09886054  1.6339210 18.421246 6.642736e-18
hp          -0.06822828  0.0101193 -6.742389 1.787835e-07
> summ$adj.r.squared
[1] 0.5891853
> summ$sigma
[1] 3.862962
```

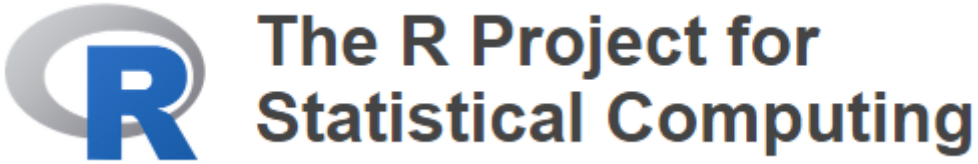


Comprehensive R Archive Network (CRAN)

- The Comprehensive R Archive Network (CRAN)
  - Repository of thousands of open source statistical/scientific R packages
  - Also includes powerful visualization/graphing packages
  - Wide selection for quant finance
  - Cutting edge statistical research often is accompanied by an R package
  - Packages published on CRAN must meet strict standards

- To download R and R packages from CRAN:

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



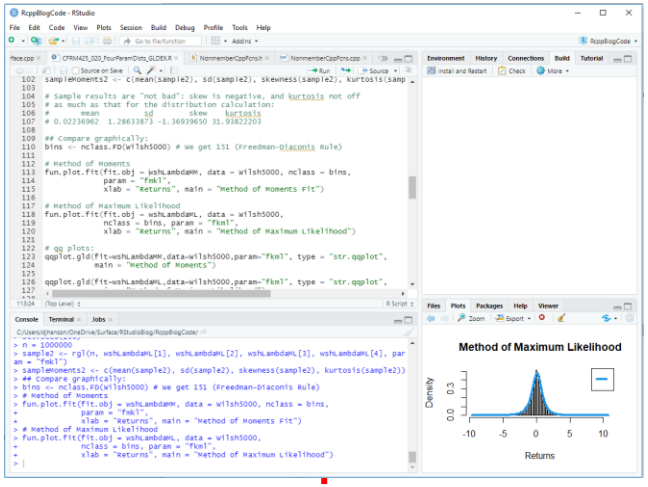
RStudio IDE

- Arguably the de facto standard IDE
- Built-in tools for building R packages
- Including integrated C++ code
- Now also supports Python

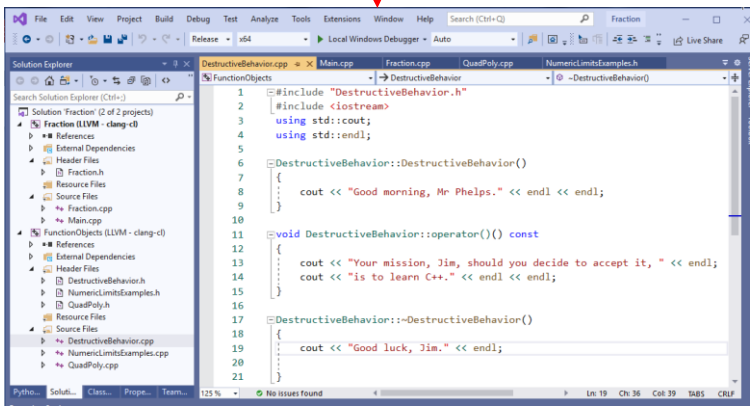
https://rstudio.com/

Call Standard/Reusable C++ from R

- R Session



- C++ Library Code



Goal

- Call R user-defined package functions in an R session…
- …that interface to standard C++
  - Higher performance
  - Reusable and consistent code (eg mathematical models libraries)
- Use the results in other R functions and visualization utilities
  - Vast set of mathematical functions in R and CRAN packages
  - Graphing capabilities in R that aren't available in C++
- Exploit the comparative advantage between R and C++
- Illustration that follows is in the RStudio IDE

Setup and Preliminaries

The Rcpp Package Solution

- The Rcpp R package* provides a reasonably painless means of creating an interface from R to C++ (especially compared to the Base R API)
- 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
- We can build the solution into our own user-defined R package
  - Compile and build it once
  - Deploy it on an arbitrary number of machines (eg in a research group, enterprise-wide, or on CRAN)

---

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

Setup: R and RStudio

- Latest version of R (4.0.3) (CRAN)

- Latest version of RStudio Desktop may be downloaded here:

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

Windows 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***:
  - Installs the gcc 8.3.0 C++ compiler on your machine (MinGW)
  - Allows you to use Rcpp on Windows
  - Download executable and follow the directions on https://cran.r-project.org/bin/windows/Rtools/
  - Caution: gcc 8.3.0 has some, but not all, C++17 features

Rcpp

- Latest stable gcc compiler version: gcc 10.2
  - C++17 full support

- Information on installation (Linux/Mac/Windows):

  https://teuder.github.io/rcpp4everyone_en/020_install.html

- Mac setup (use the most recent Clang compiler in Xcode):

  https://thecoatlessprofessor.com/programming/cpp/r-compiler-tools-for-rcpp-on-macos/

Setup: Installation of Rcpp

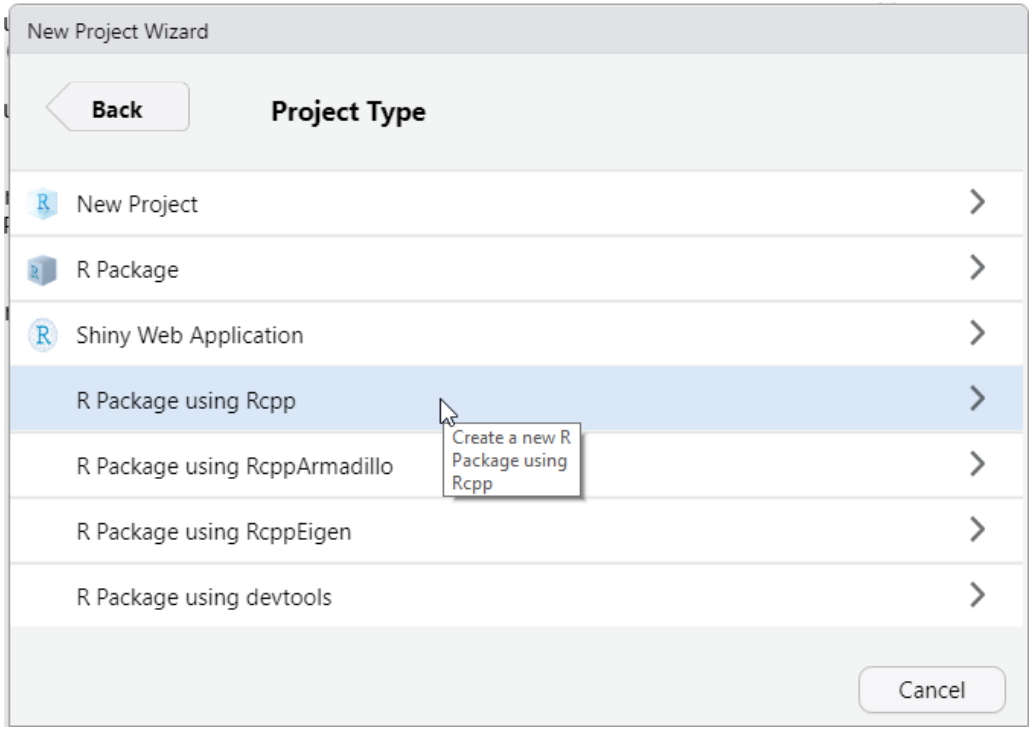
- After installing a compatible compiler, open RStudio, and install the **`Rcpp`** package from CRAN
  - Either run the following R command:

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

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

Creating an Rcpp Package Project in RStudio

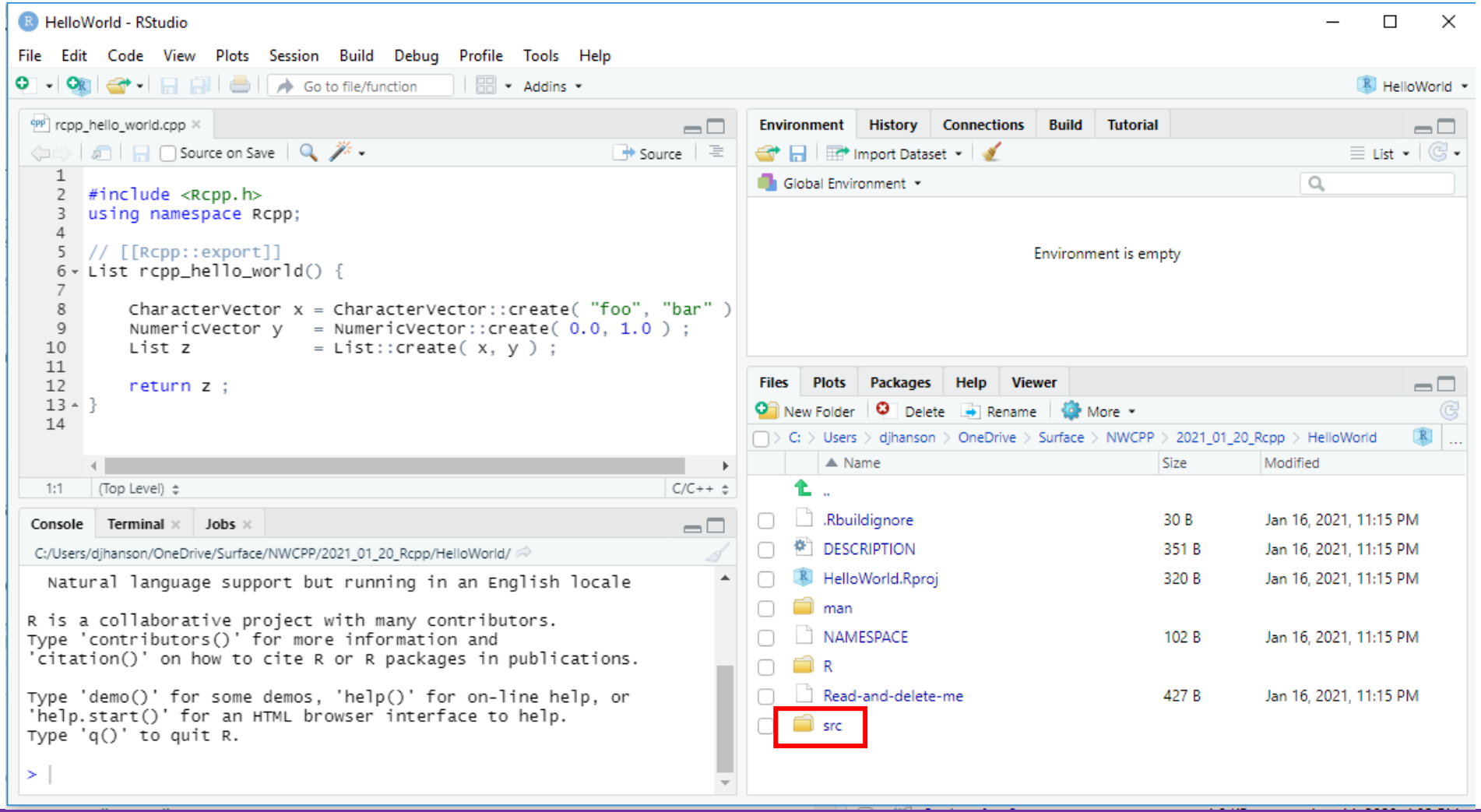
- Select ***File*/*New Project*/*New Directory*/*R Package*** using Rcpp



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

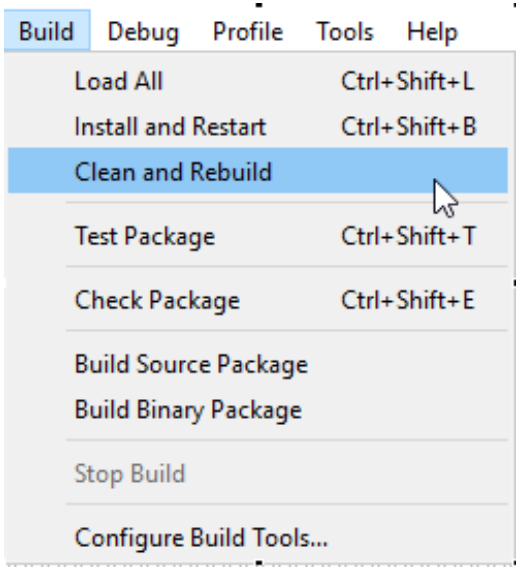
Creating an Rcpp Package Project in RStudio

- A new Rcpp project is created
- C++ files go in the `src` subdirectory
- A sample `rcpp_hello_world.cpp` example is provided by default

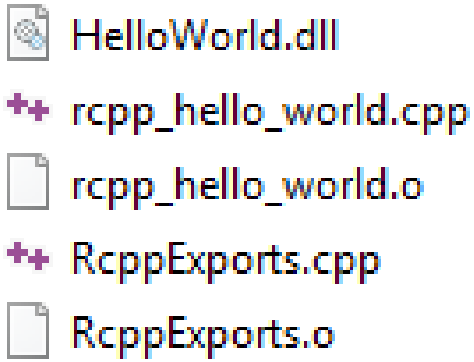


Build the "Hello World" Rcpp Package Project in RStudio

- 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):



- Similar for a "real" project

Integrating Standard and Reusable C++ Code

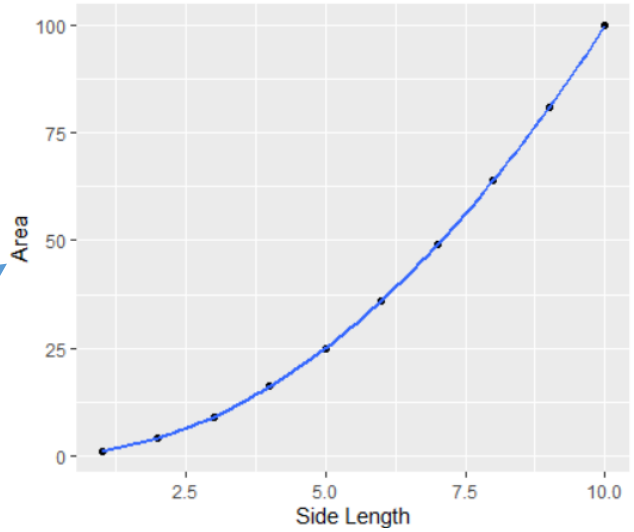
"I'm firmly convinced that behind every great man is a great computer."

Setting up an Interface to Reusable C++ Code

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

R Session

```
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 (Rcpp)

Non-member interface functions exported to R

Independent of reusable C++ code

Mathematical models code base

Standard C++ ONLY

classes

non-member functions

C++ Standard Library (gcc)

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 C++ functions and classes that facilitate the interface with R
  - **`#include<Rcpp.h>`**
  - **`Rcpp`** namespace

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

- We will only need one class from Rcpp, an STL-compliant container class that emulates an R numeric vector in C++
  - **`Rcpp::NumericVector`**

- Plus two Rcpp C++ functions:

```
// Transfer data from NumericVector to std::vector<double>
Rcpp::as<std::vector<double>>(.)

// Reassign the results from a vector<double>
// to an Rcpp::NumericVector:
v = Rcpp::wrap(.);
```

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

- A C++ interface function exported to R is indicated by the tag above its signature:

```
// [[Rcpp::export]]
double fcn(Rcpp::NumericVector v)
{
    auto w = Rcpp::as<std::vector<double>>(v);

    // Call function in standard C++ code base:
    double y = doSomething(w);

    return y;
}
```

C++ Interface Layer (Rcpp)
Non-member interface functions exported to R

Independent of reusable C++ code

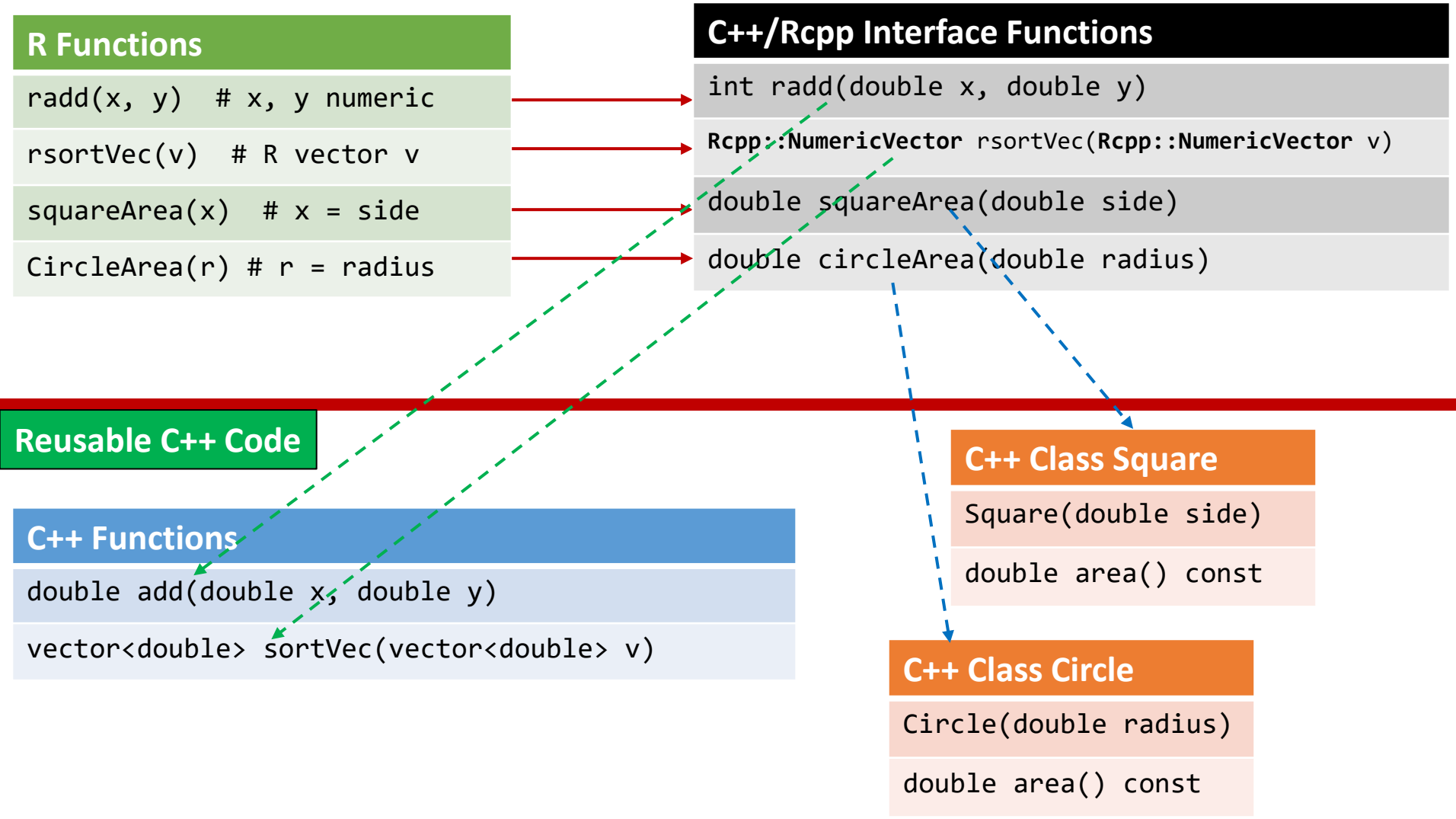
Mathematical models code base
Standard C++ ONLY

classes
non-member functions

- **NOTE: *This is at the interface level –*** We can avoid polluting the standard C++ code base with the **Rcpp** declaration or namespace

Case w/o external C++ libraries

- *Prime Directive*: Keep reusable C++ code ***standard*** and ***independent*** of Rcpp interface



Example in RStudio: Interface Files

- Write the C++ interface files:
  - **`#include`** the declaration files from the reusable C++ code base
    - Instantiate Square and Circle objects 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

Sort an R Numeric Vector

- Rcpp interface

  This function is exported to R

```cpp
#include "NonmemberCppFcns.h"
#include <vector>
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::NumericVector rSortVec(Rcpp::NumericVector v)
{
  // Transfer data from NumericVector to std::vector<double>
  auto stlVec = Rcpp::as<std::vector<double>>(v);

  // Call the reusable sortVec(.) function, with the expected
  // std::vector<double> argument:
  stlVec = sortVec(stlVec);

  // Reassign the results from the vector<double> return object
  // to the same NumericVector v, using Rcpp::wrap(.):
  v = Rcpp::wrap(stlVec);

  // Return as an Rcpp::NumericVector:
  return v;
}
```

- Reusable C++ functions

```cpp
#include "NonmemberCppFcns.h"
#include <algorithm>
#include <numeric>
#include <cmath>

using std::vector;

double add(double x, double y)
{
  return x + y;
}

vector<double> sortVec(vector<double> v)
{
  sort(v.begin(), v.end());
  return v;
}
```

Compute the Area of a Square and Circle

- Rcpp interface

  These functions are exported to R

- Reusable C++ classes

```cpp
#include "ConcreteShapes.h"
// [[Rcpp::export]]
double squareArea(double side)
{
  Square sq(side);
  return sq.area();
}

// Interface to Circle member
// function area(.):
// [[Rcpp::export]]
double circleArea(double radius)
{
  Circle circ(radius);
  return circ.area();
}
```

```cpp
#include "ConcreteShapes.h"

// Circle implementation:
Circle::Circle(double radius) :radius_(radius) {}
double Circle::area() const
{
  double pi = std::acos(-1.0);
  return pi * radius_ * radius_;
}

// Square implementation:
Square::Square(double side) :side_(side) {}
double Square::area() const
{
  return side_ * side_;
}
```

In an R Session (Console)

- Load the R package

```
library(RcppBlogCode)
```

- Sort a vector

```
(x <- c(5:1))
rSortVec(x)
```

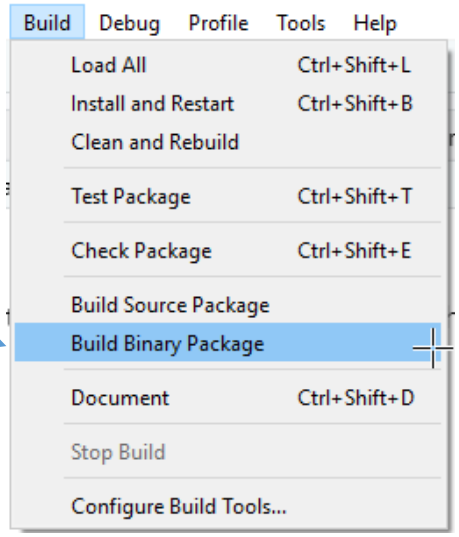
- Calculate the area of a square and a circle

```
squareArea(4)
circleArea(1)
```

- These are R functions
  - Behind the scenes, they call reusable C++ code
  - The user doesn't need to know or care

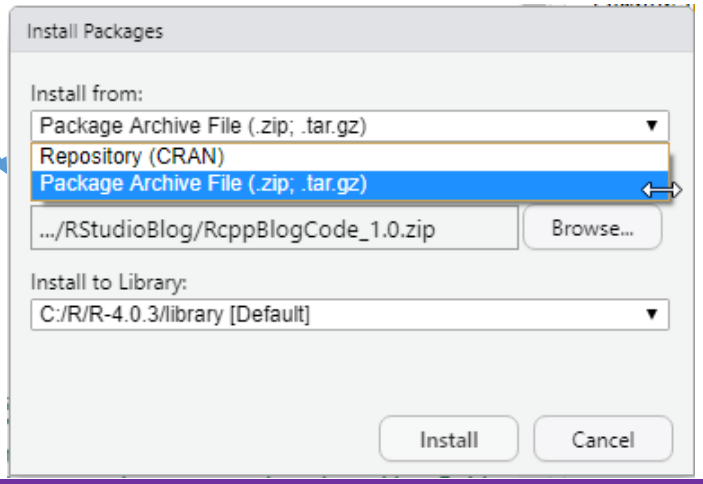
Distributing the Package

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



  - This will generate
    - **`RcppBlogCode_1.0.zip`** on Windows
    - **`RcppBlogCode_1.0.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(RcppBlogCode_1.0)`** in the local R session to load it
    - Call the functions

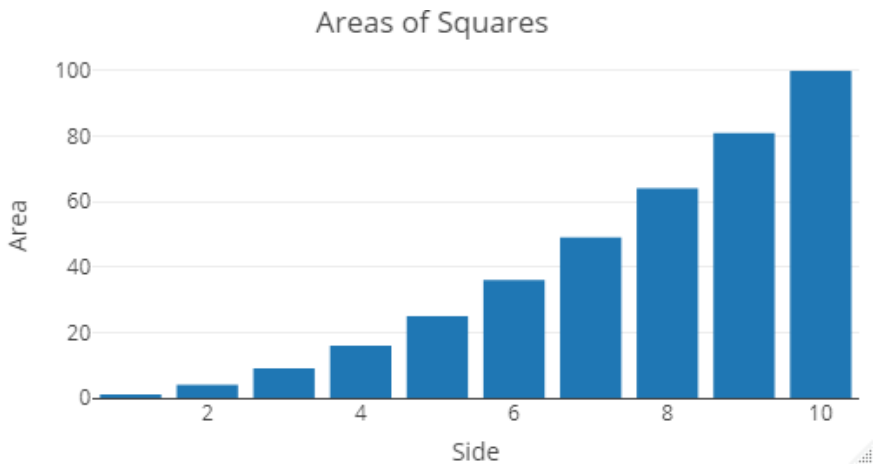


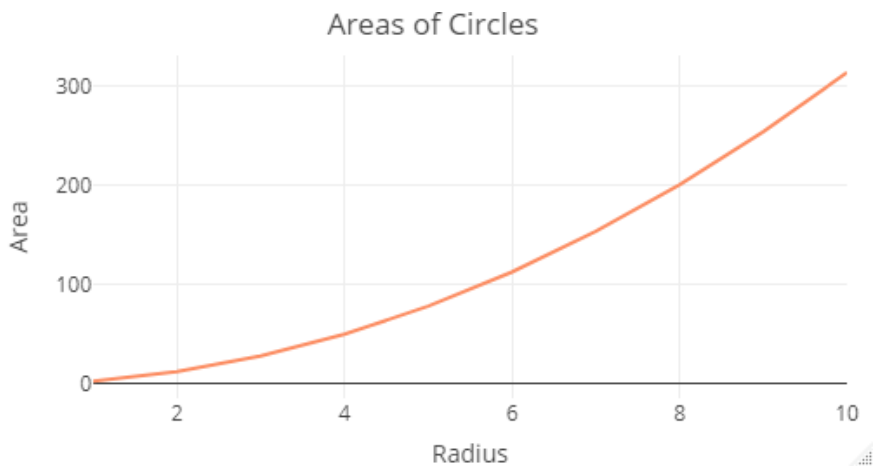
In an R Session

- A little more interesting, use the *plotly* package in R (CRAN)
- Create R numeric vectors of square sides and radii, and calculate respective areas with the vectorizing **`sapply(.)`** function:

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

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

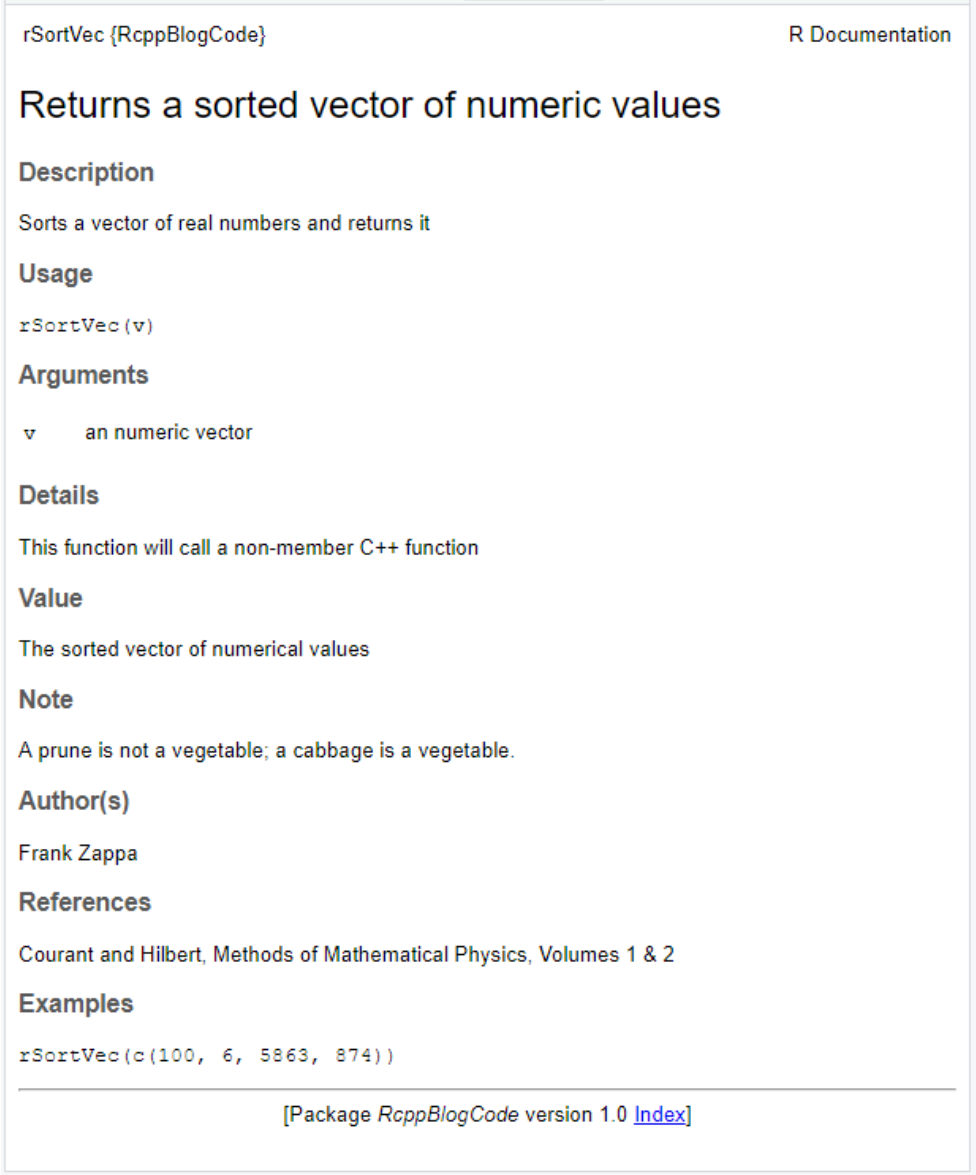
- Apply the **`plot_ly(.)`** function (see sample code):





Example/Documentation

- Building the package also generates standard CRAN-compliant documentation for the package using RStudio utilities



rSortVec {RcppBlogCode} R Documentation

Returns a sorted vector of numeric values

Description

Sorts a vector of real numbers and returns it

Usage

```
rSortVec(v)
```

Arguments

v	an numeric vector

Details

This function will call a non-member C++ function

Value

The sorted vector of numerical values

Note

A prune is not a vegetable; a cabbage is a vegetable.

Author(s)

Frank Zappa

References

Courant and Hilbert, Methods of Mathematical Physics, Volumes 1 & 2

Examples

```
rSortVec(c(100, 6, 5863, 874))
```

[Package *RcppBlogCode* version 1.0 Index]

- Requires updating and formatting an Rd file for each function

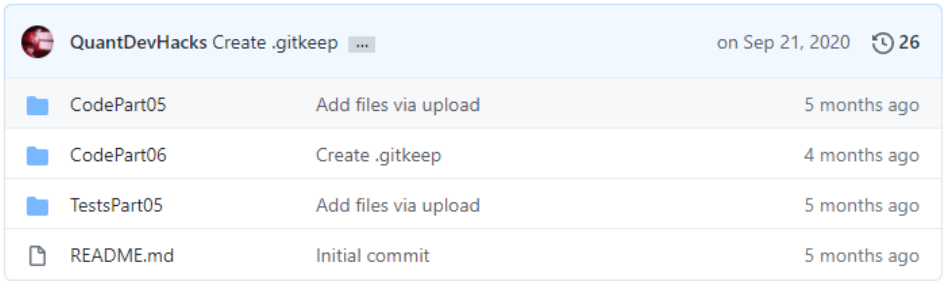
More Information (Shameless Plug)

- Six-part blog series published on RStudio *R Views*:

https://rviews.rstudio.com/tags/rcpp/

- Accompanying source code:

https://github.com/QuantDevHacks/RcppBlogCode



R Package Integration with Modern Reusable C++ Code Using Rcpp - Part 6
Daniel Hanson 2020-09-28

R Package Integration with Modern Reusable C++ Code Using Rcpp - Part 5
Daniel Hanson 2020-08-24

R Package Integration with Modern Reusable C++ Code Using Rcpp - Part 4
Daniel Hanson 2020-08-18

R Package Integration with Modern Reusable C++ Code Using Rcpp - Part 3
Daniel Hanson 2020-07-31

R Package Integration with Modern Reusable C++ Code Using Rcpp - Part 2
Daniel Hanson 2020-07-14

R Package Integration with Modern Reusable C++ Code Using Rcpp
Daniel Hanson 2020-07-08

Thank You

- Contact:
  - hansondj@uw.edu
  - https://www.linkedin.com/in/danielhanson/

Questions?

-